

# A MODERN TESTING TALE





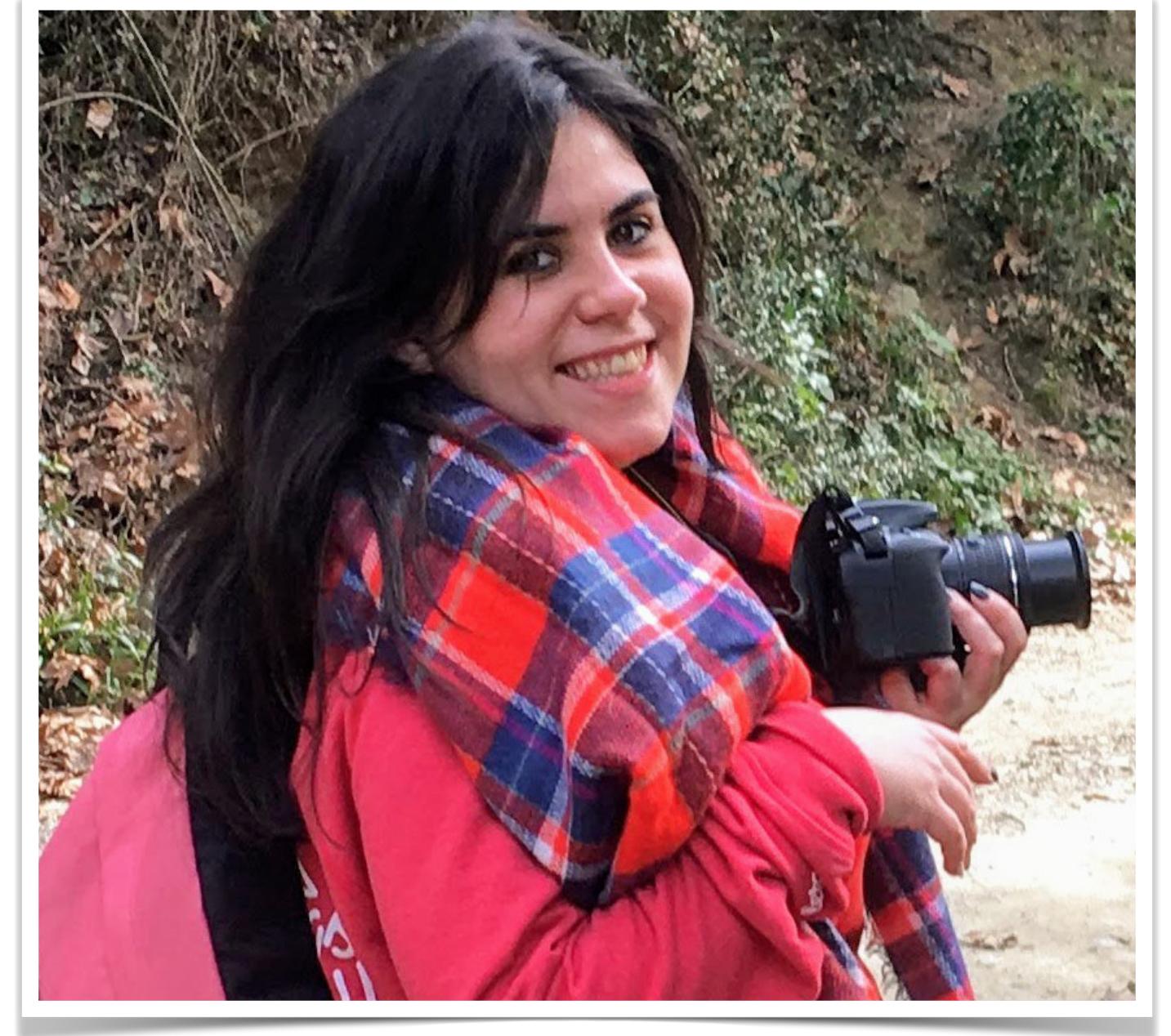


will AI free us from doing unit  
tests?



# Isabel Garrido

- Senior Backend developer
- More Than 10 years of experience
- Jetbrains Community Contributor
- Former testing teacher at laSalle university
- Online Kotlin course instructor



A photograph of a person from behind, standing on a dark asphalt road that curves through a landscape of green and yellow rolling hills under a clear blue sky.

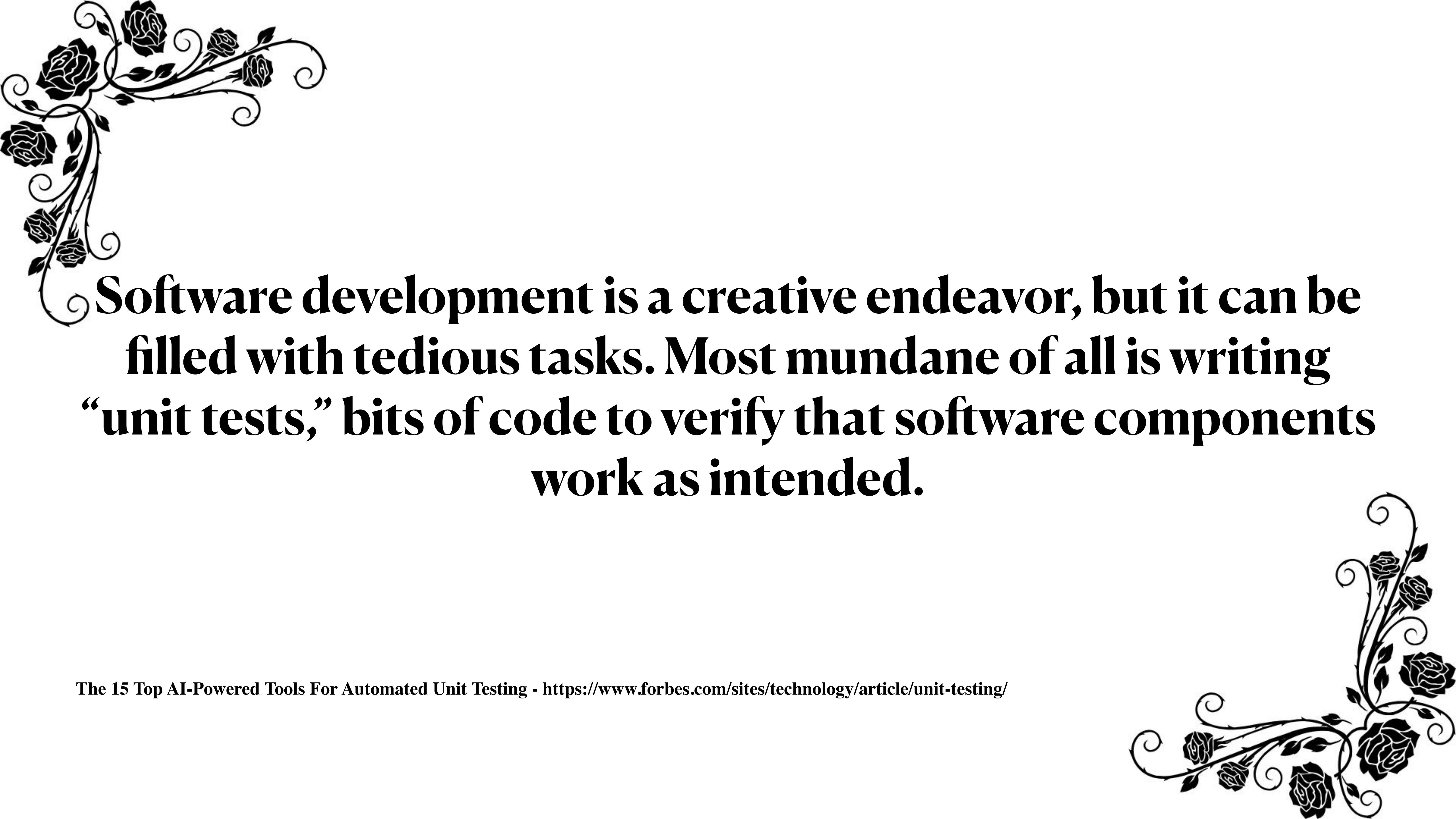
# The beginning

<https://www.freepik.com/author/wirestock>



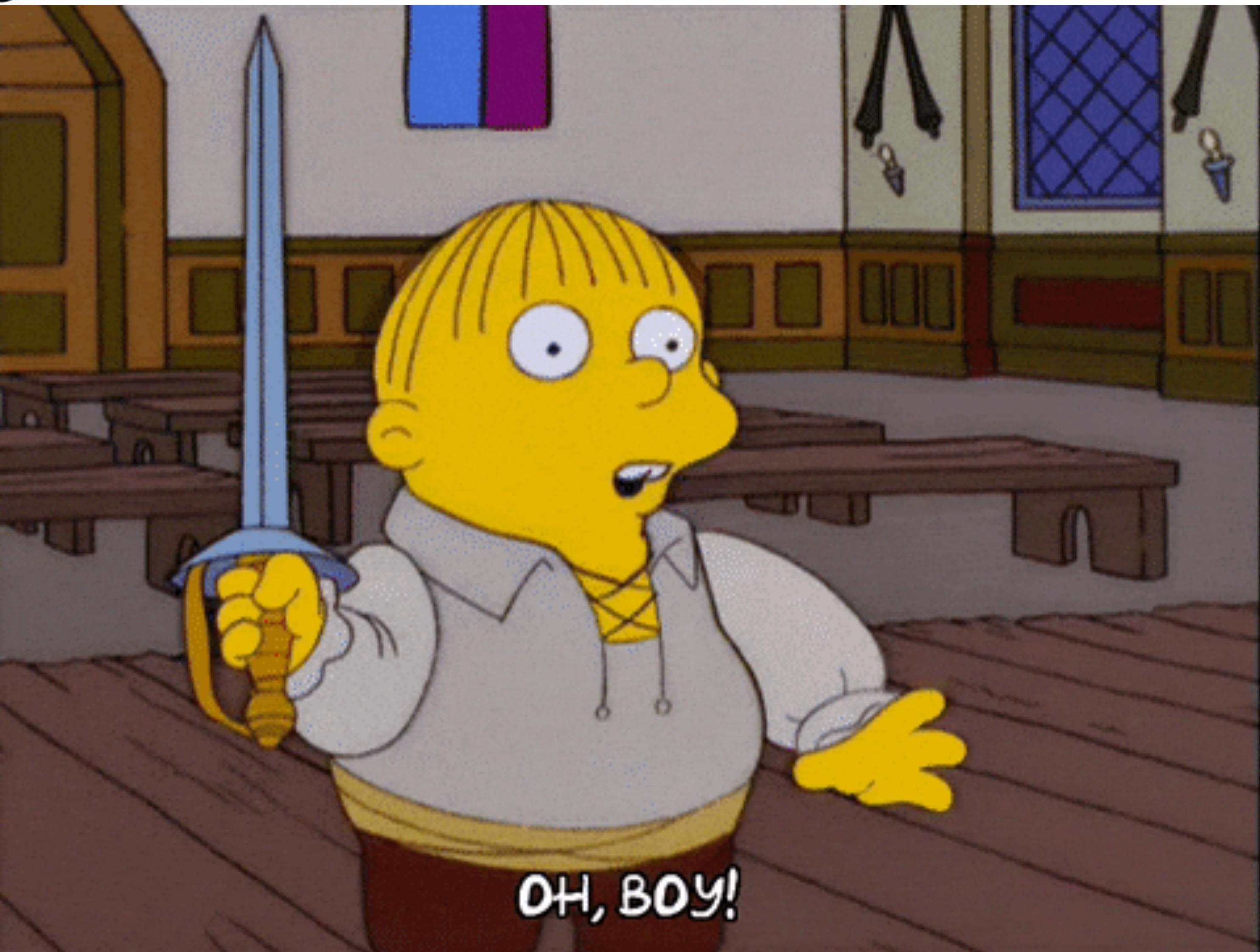
# **Why are you doing tests instead of developing other features? It's a waste of your time**

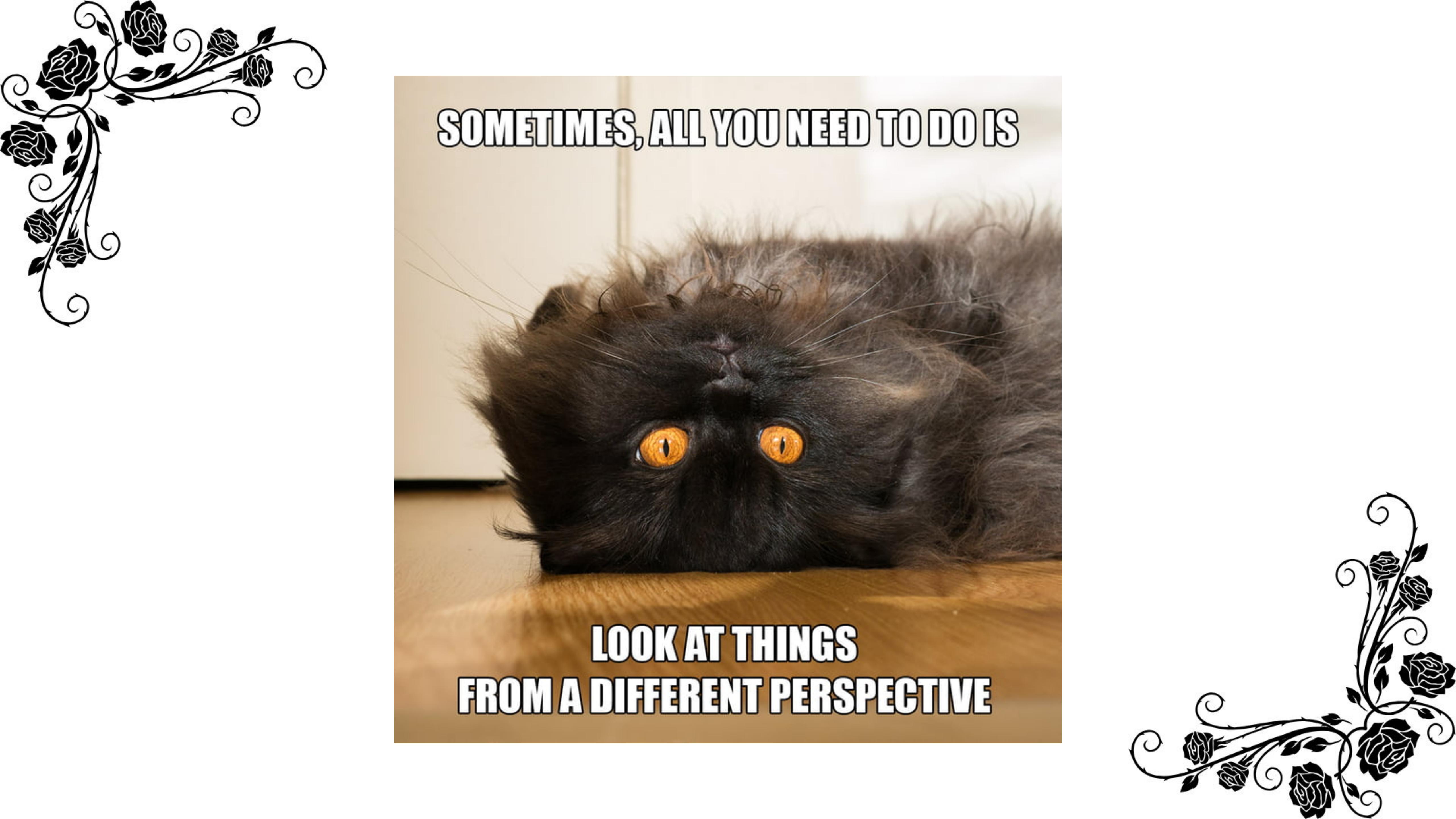
My friend during our trip



**Software development is a creative endeavor, but it can be filled with tedious tasks. Most mundane of all is writing “unit tests,” bits of code to verify that software components work as intended.**

The 15 Top AI-Powered Tools For Automated Unit Testing - <https://www.forbes.com/sites/technology/article/unit-testing/>



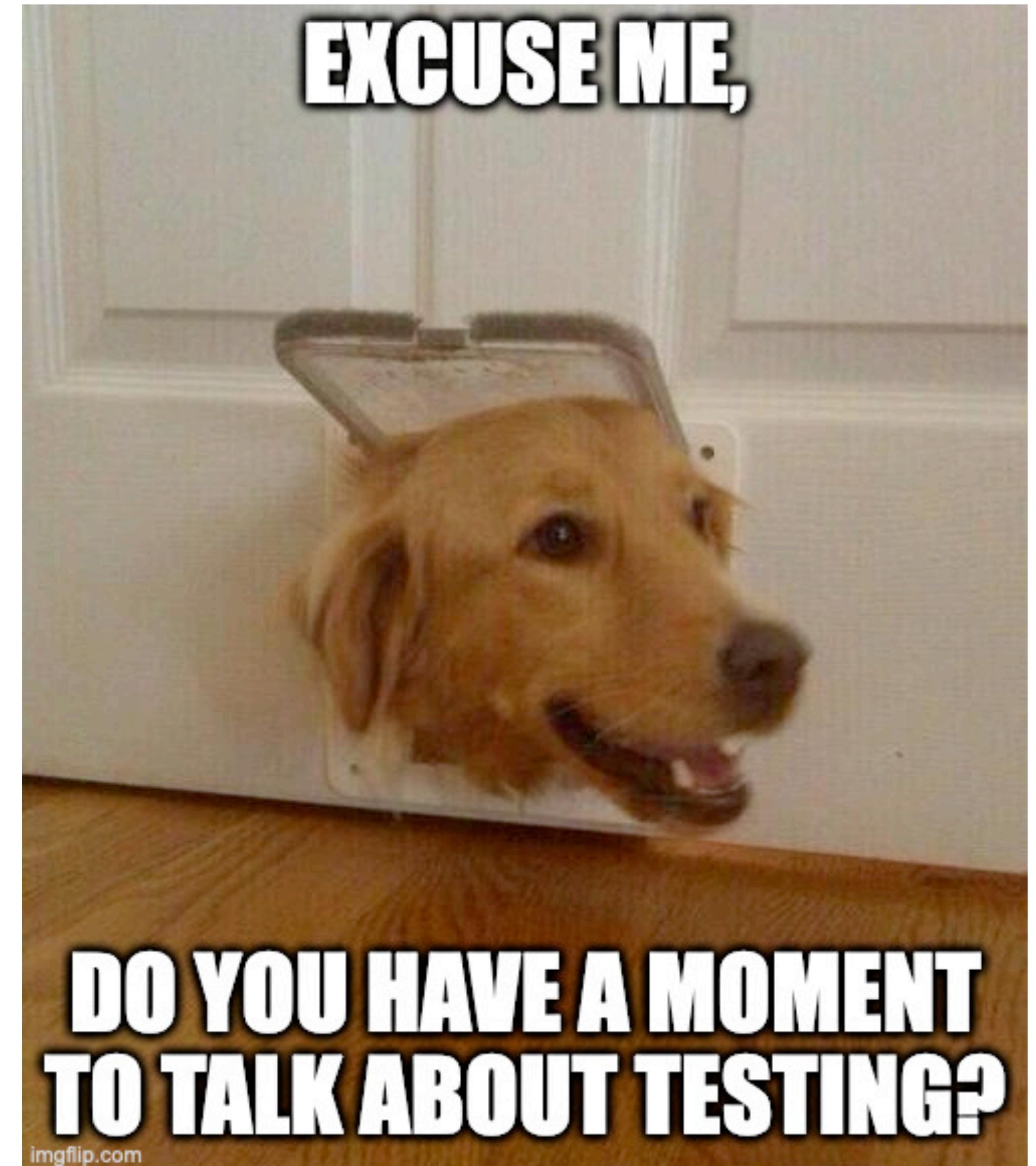


**SOMETIMES, ALL YOU NEED TO DO IS**



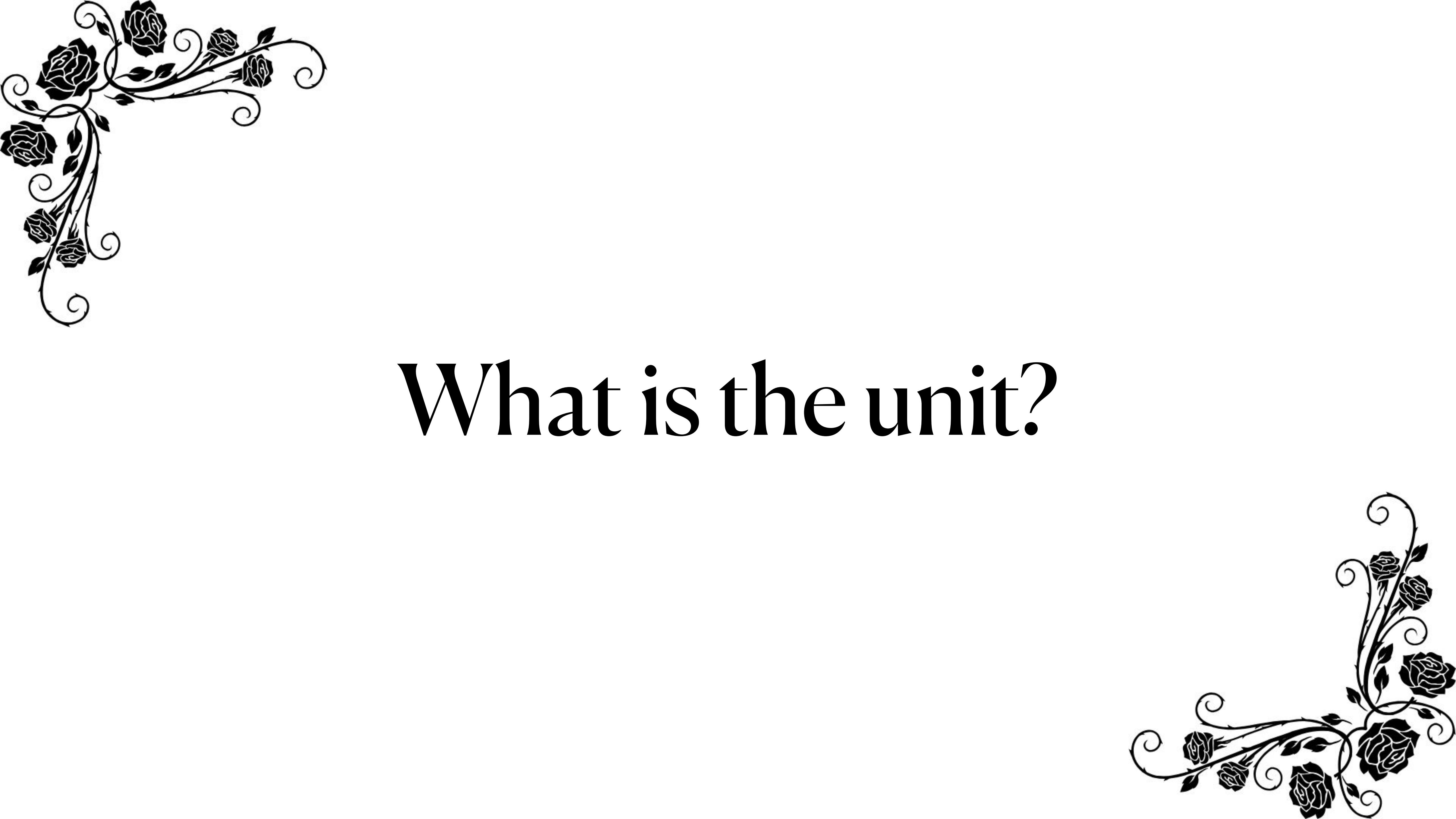
**LOOK AT THINGS  
FROM A DIFFERENT PERSPECTIVE**





imgflip.com

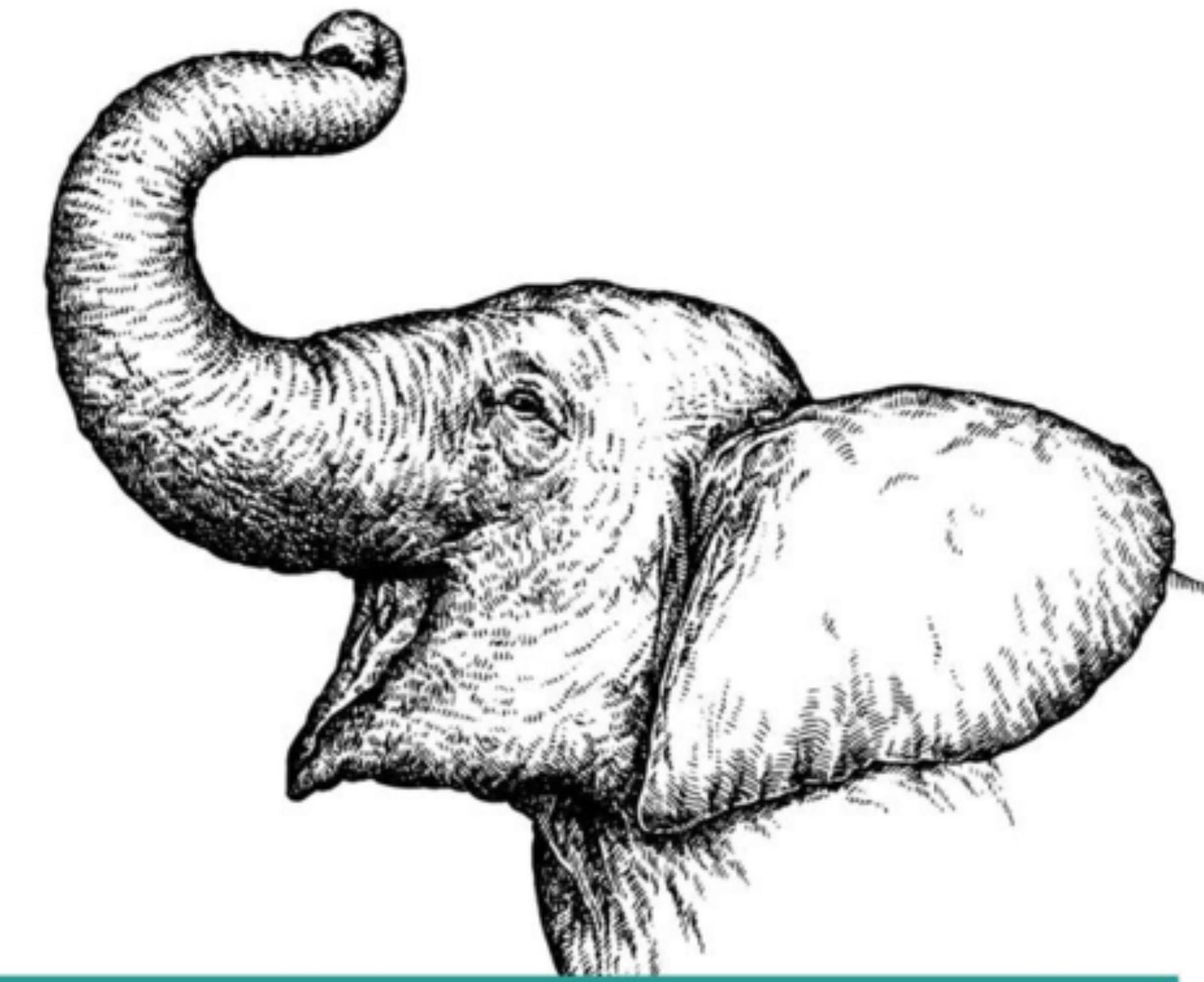




What is the unit?



*The answer to every programming question ever conceived*



# It Depends

*The Definitive Guide*

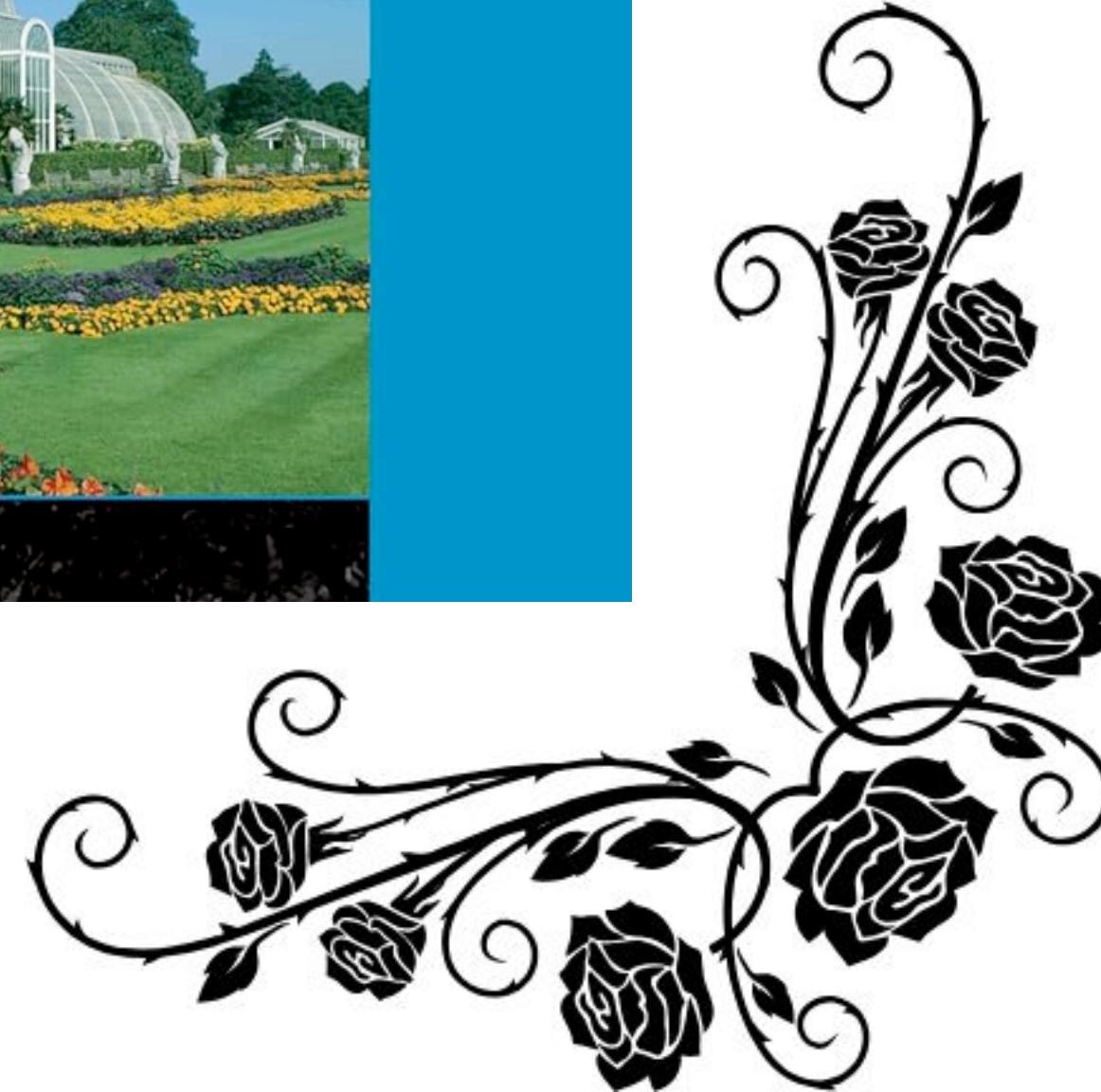
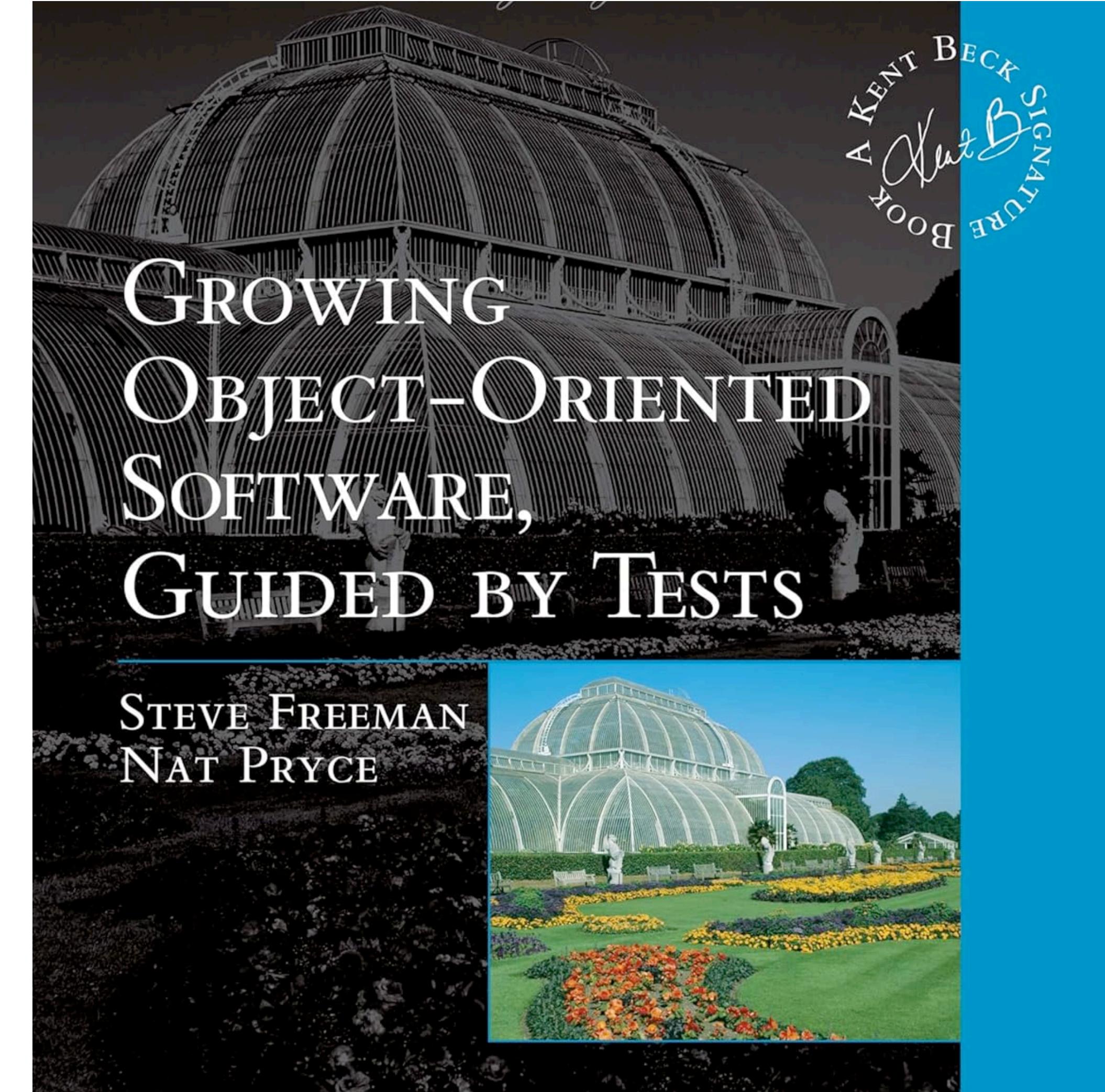
O RLY?

@ThePracticalDev



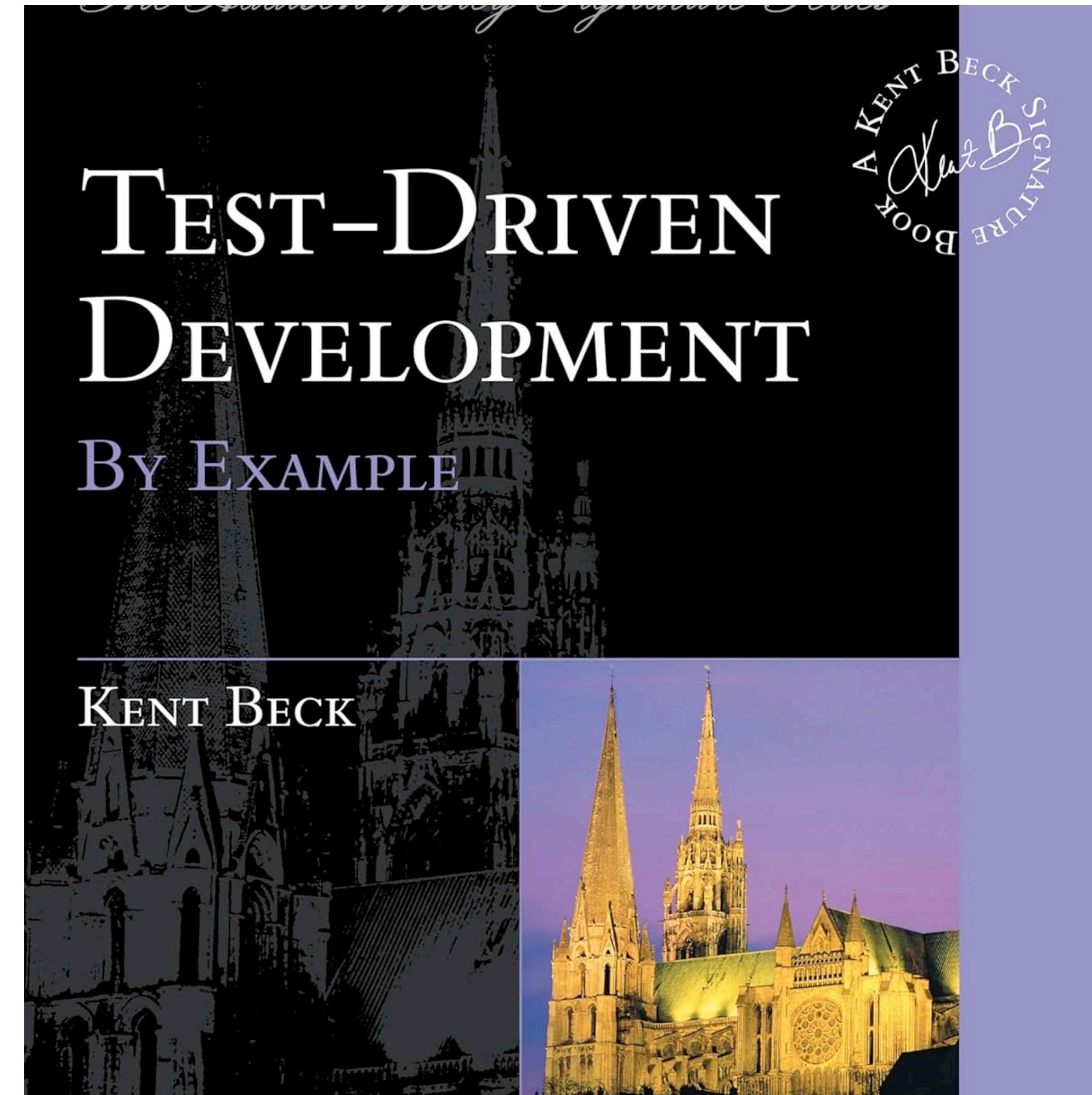
# London School

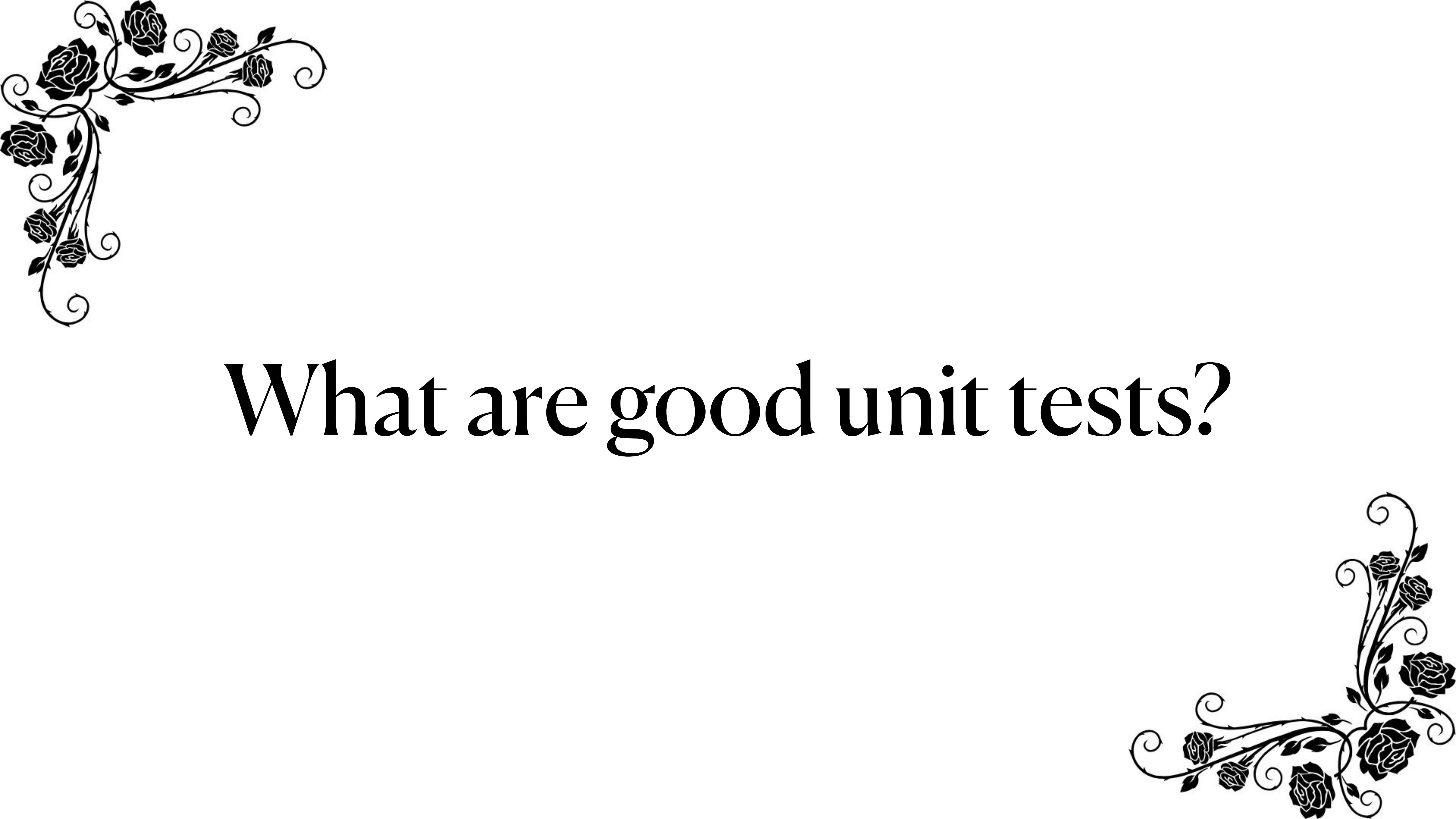
- The unit is a class
- From outside to inside
- Focus on interaction



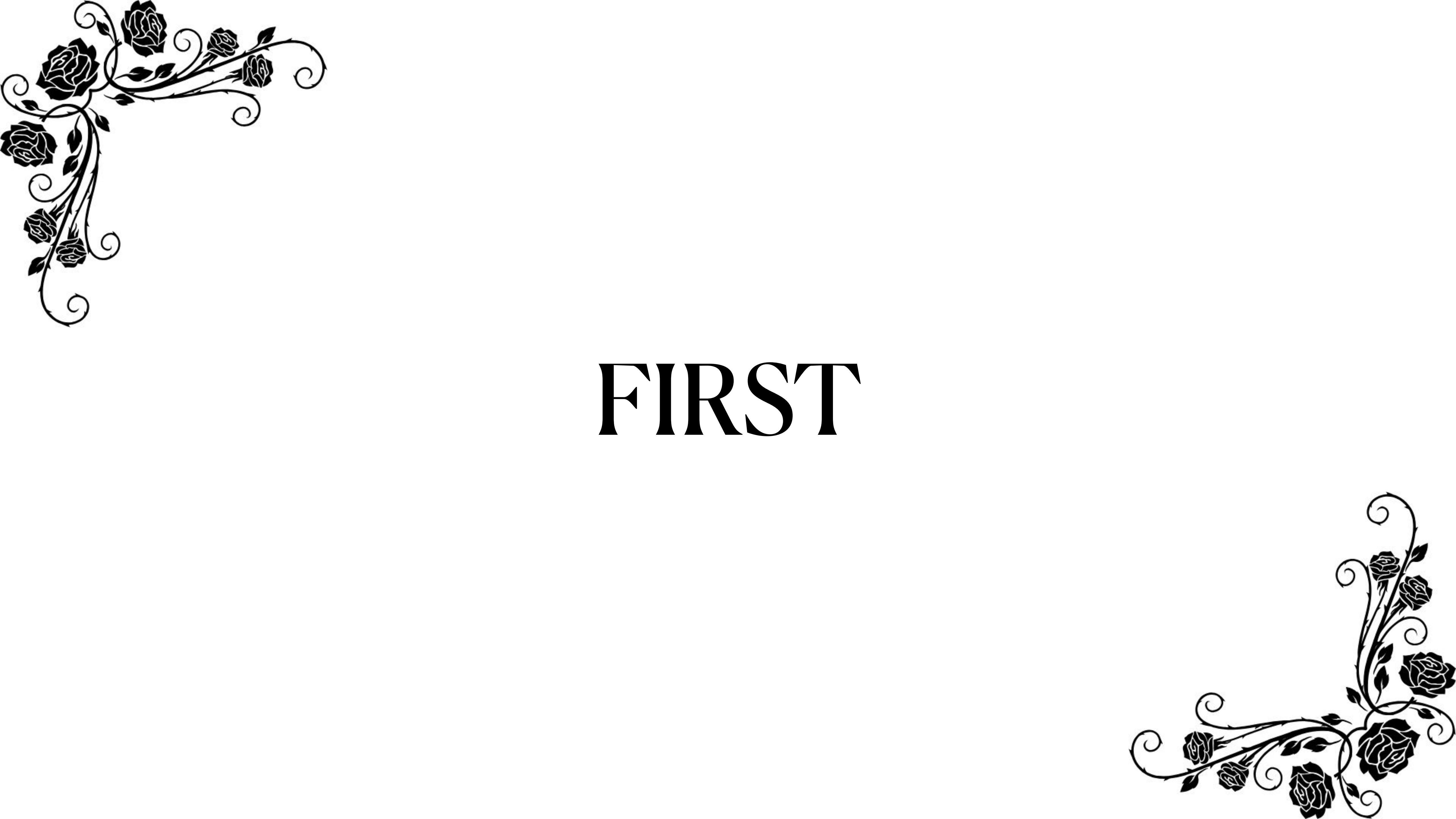
# Chicago School

- The unit is a small collection of pieces that together delivers the behavior we want
- From inside to outside



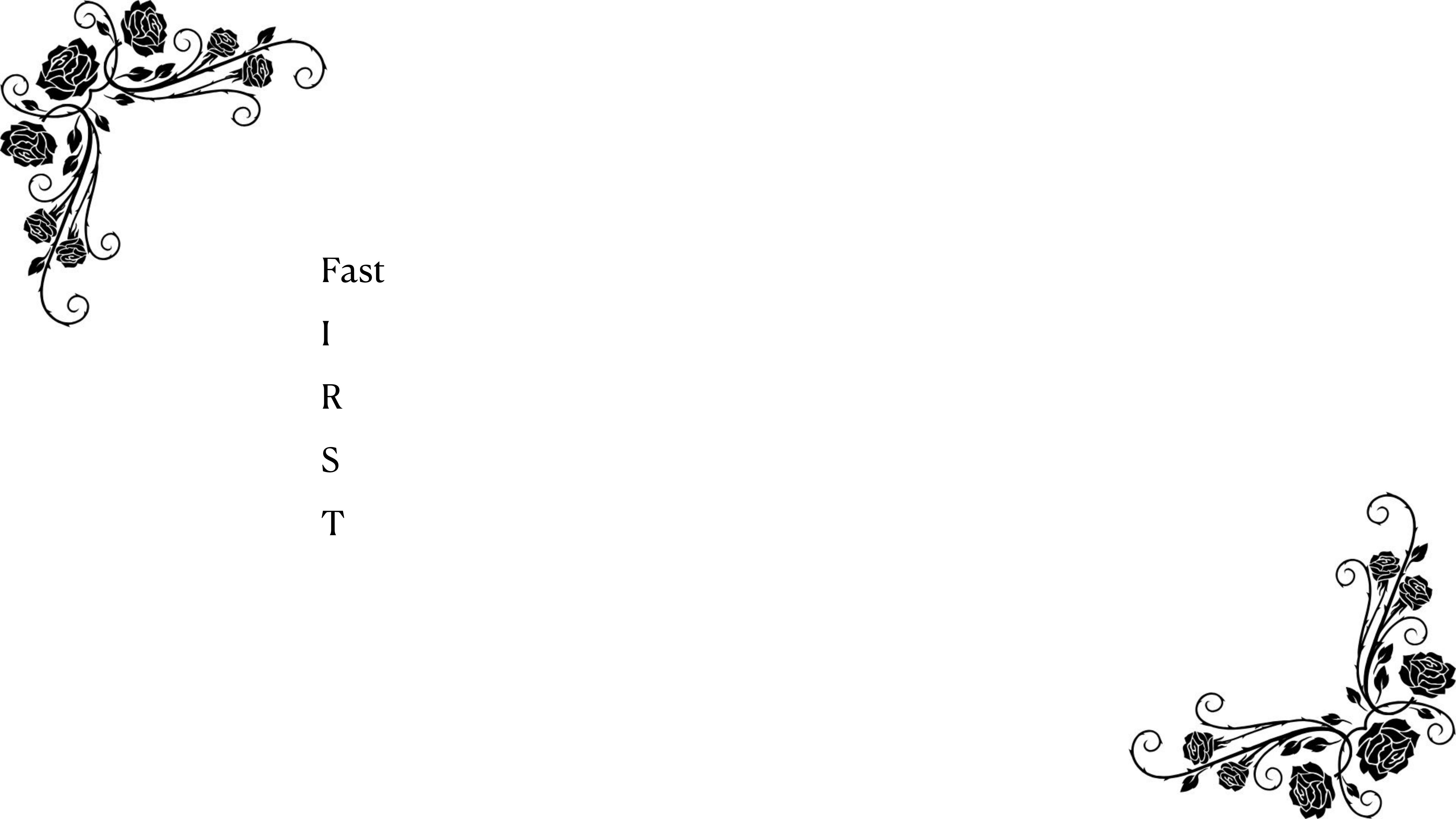


What are good unit tests?



The image features a central decorative floral border composed of black outlines on a white background. The border is symmetrical and includes several rose flowers, buds, and swirling vines in each corner.

FIRST



Fast  
I  
R  
S  
T



Fast  
Isolated  
R  
S  
T





Fast  
Isolated  
Repeatable

S

T

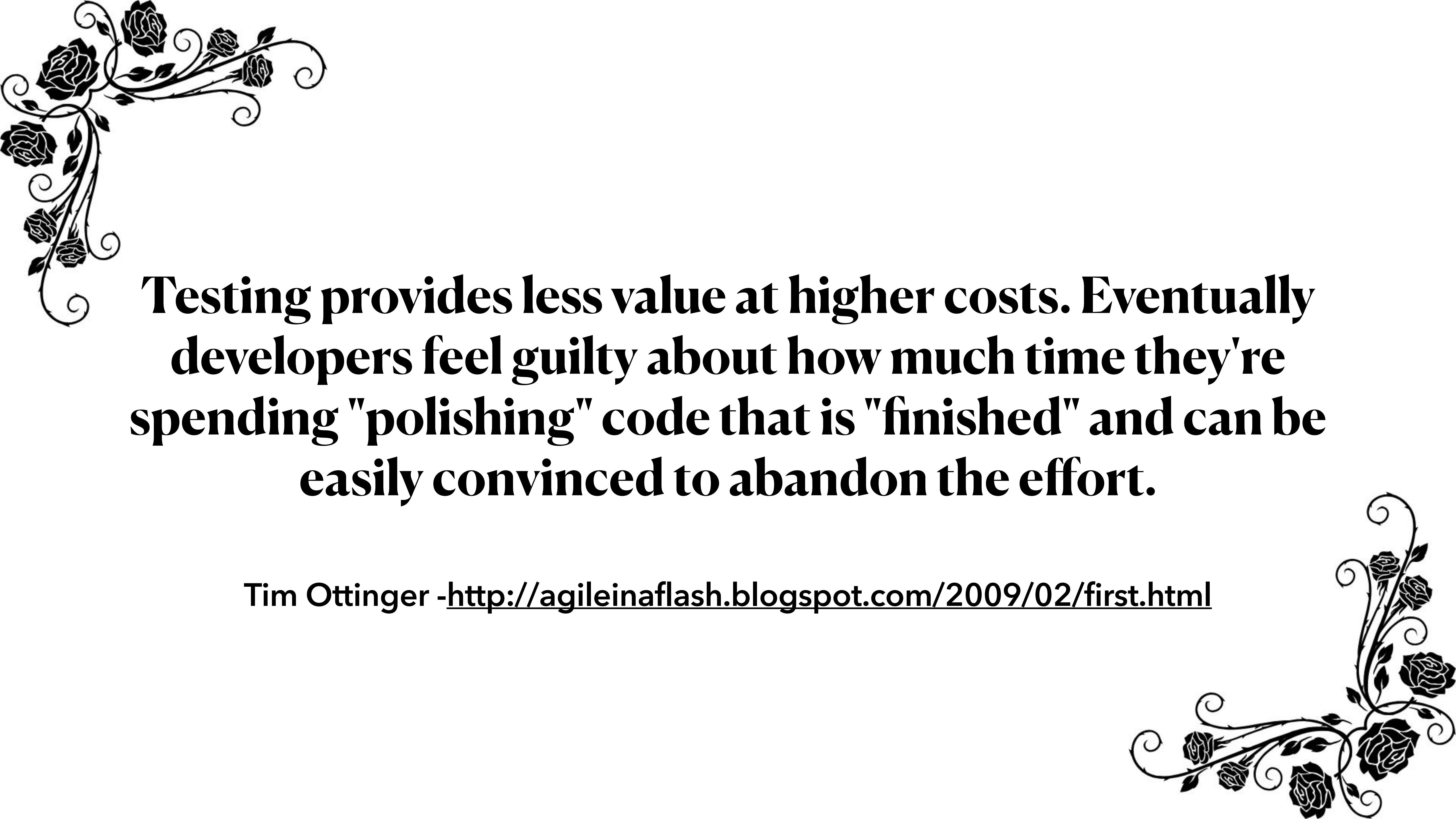




Fast  
Isolated  
Repeatable  
Self-validating  
T



Fast  
Isolated  
Repeatable  
Self-validating  
Timely

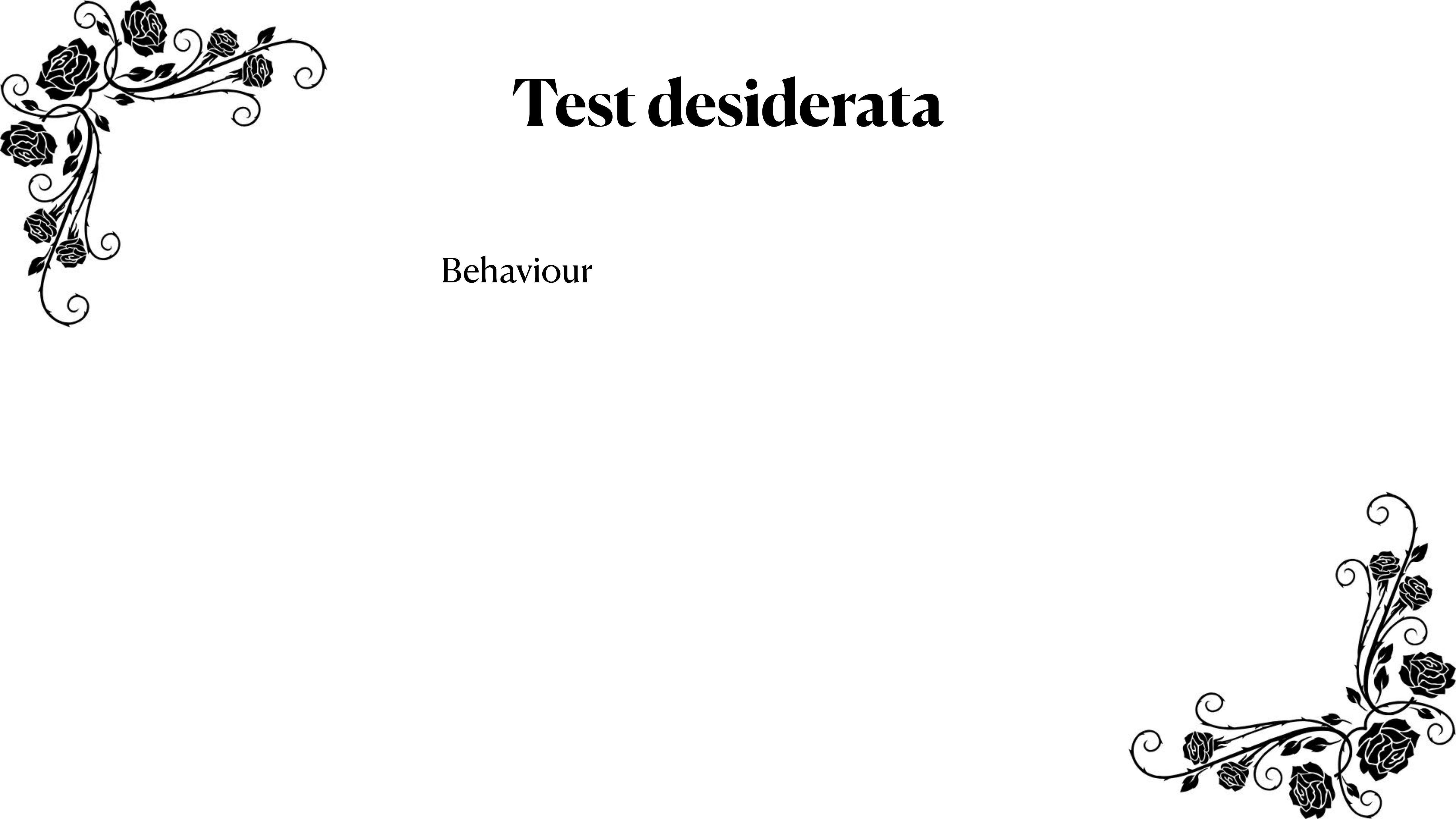


**Testing provides less value at higher costs. Eventually developers feel guilty about how much time they're spending "polishing" code that is "finished" and can be easily convinced to abandon the effort.**

Tim Ottinger -<http://agileinaflash.blogspot.com/2009/02/first.html>

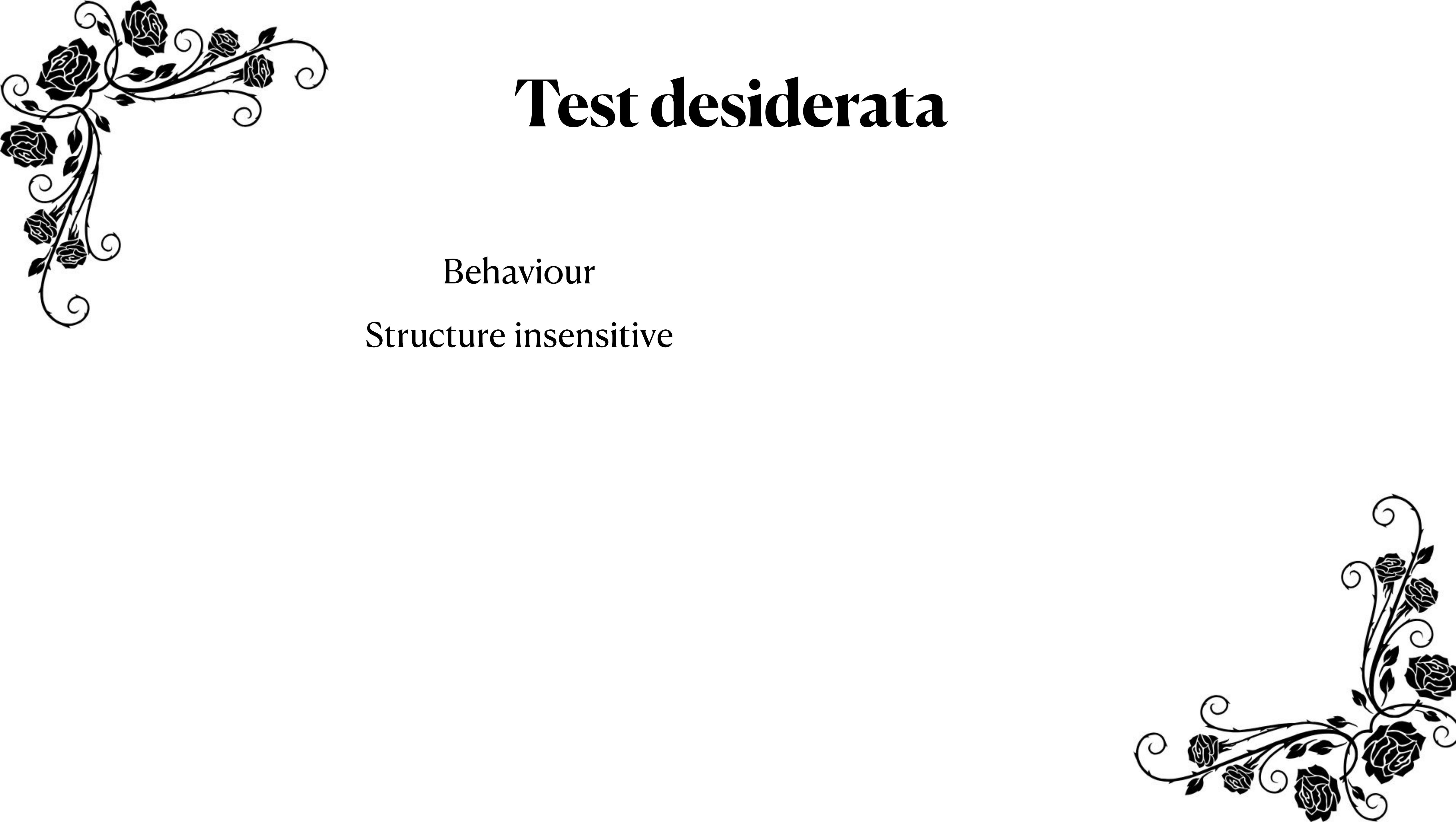


# Test Desiderata



# **Test desiderata**

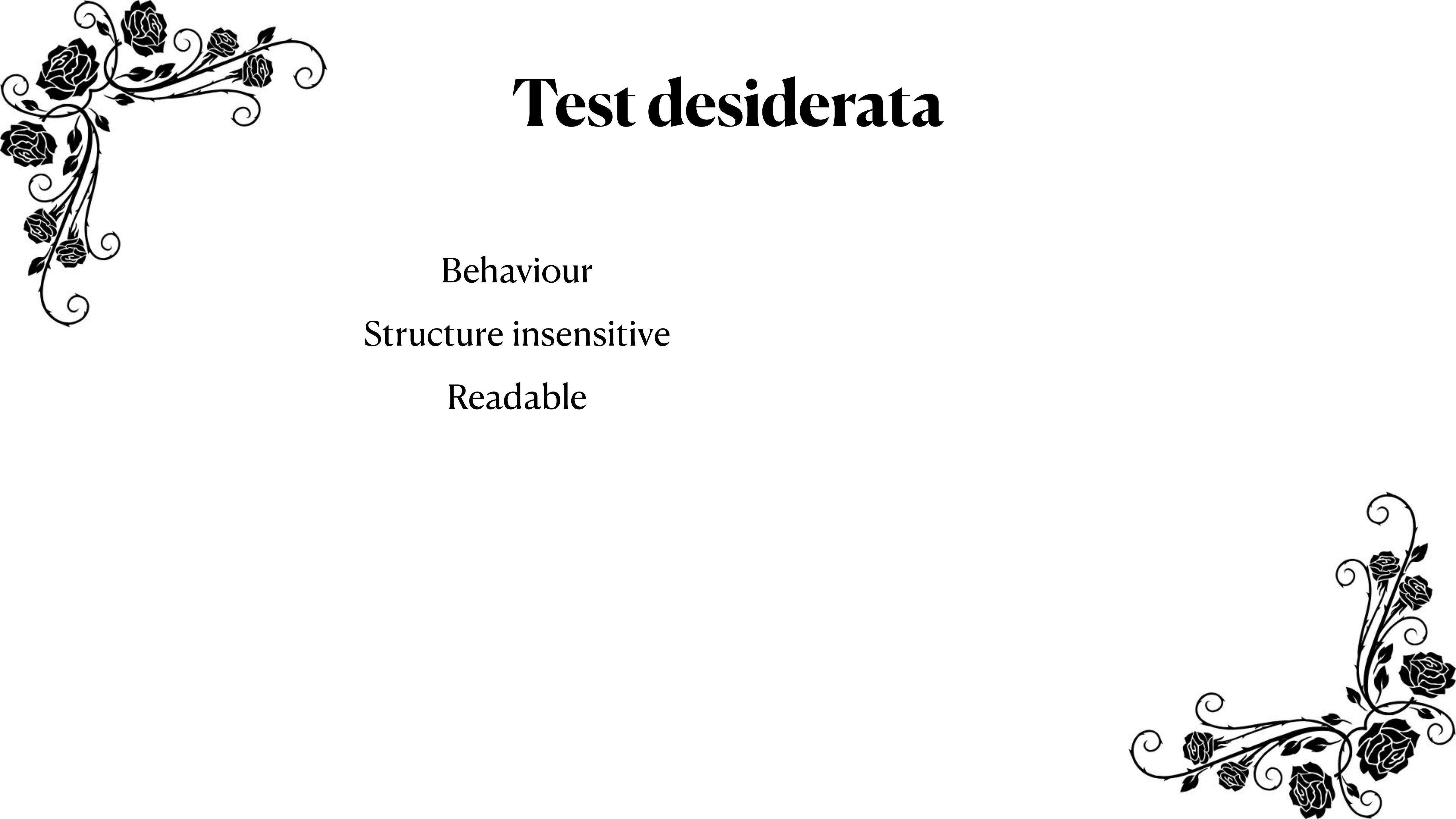
Behaviour



# **Test desiderata**

Behaviour

Structure insensitive

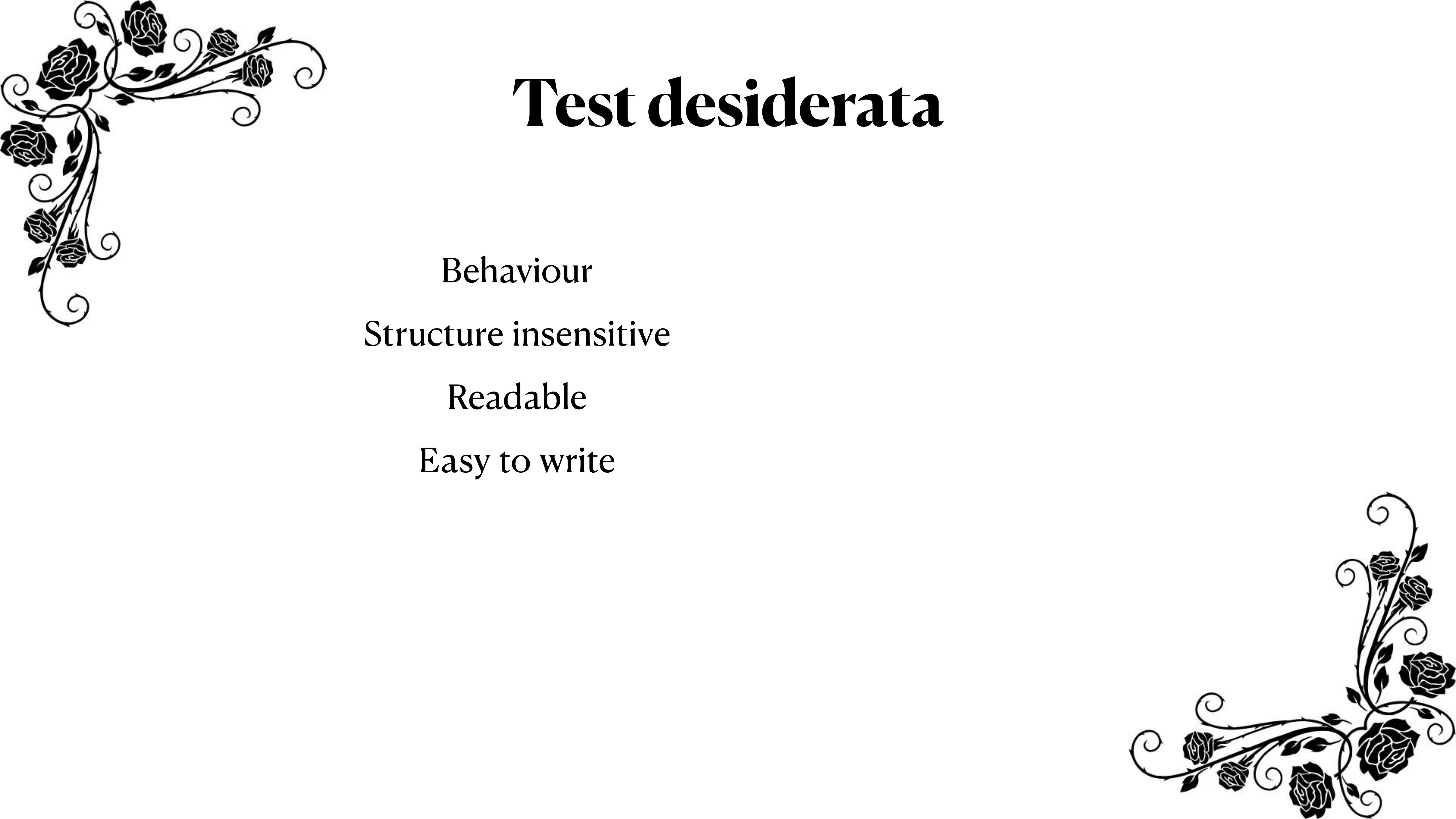


# **Test desiderata**

Behaviour

Structure insensitive

Readable



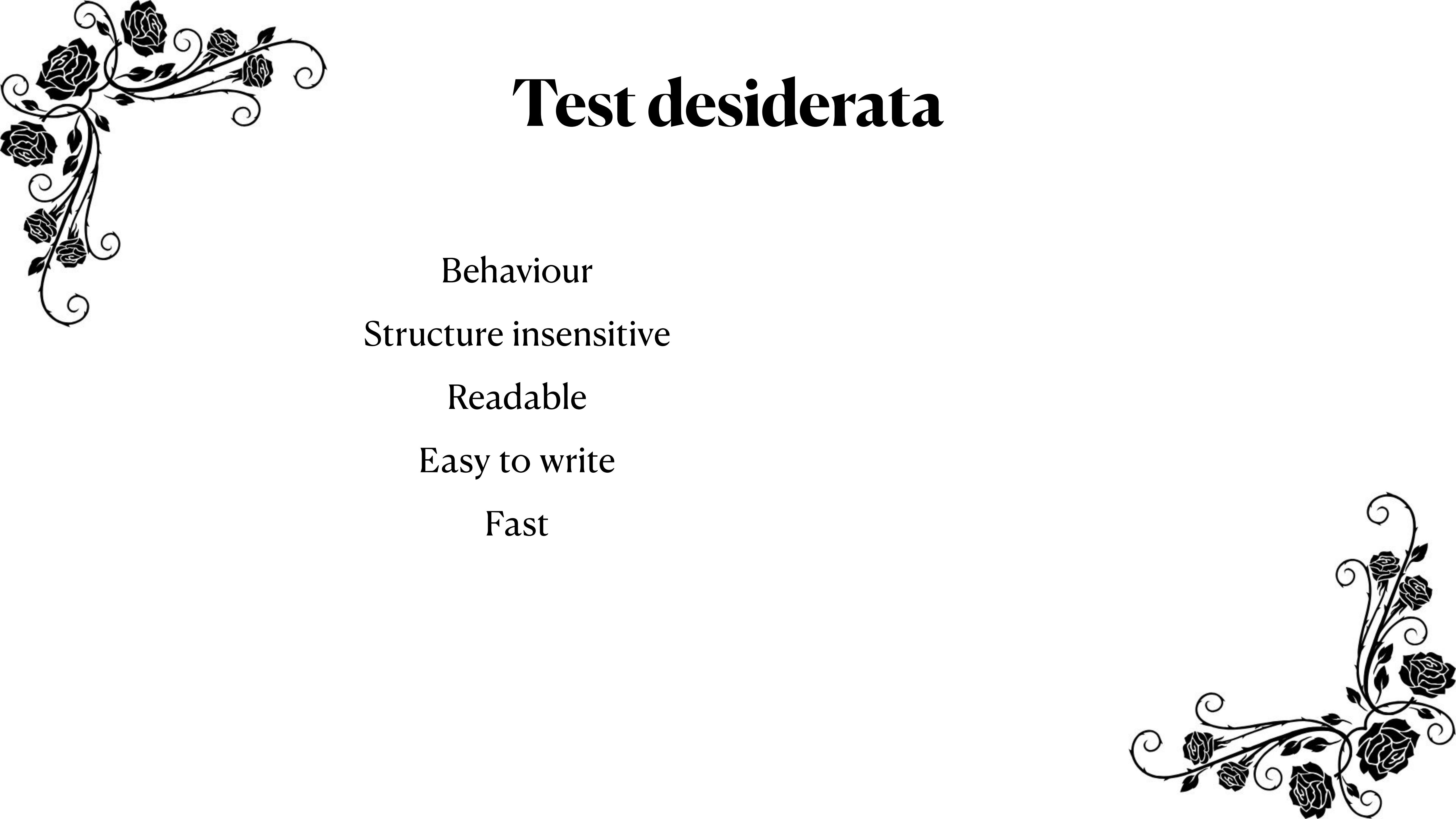
# Test desiderata

Behaviour

Structure insensitive

Readable

Easy to write



# Test desiderata

Behaviour

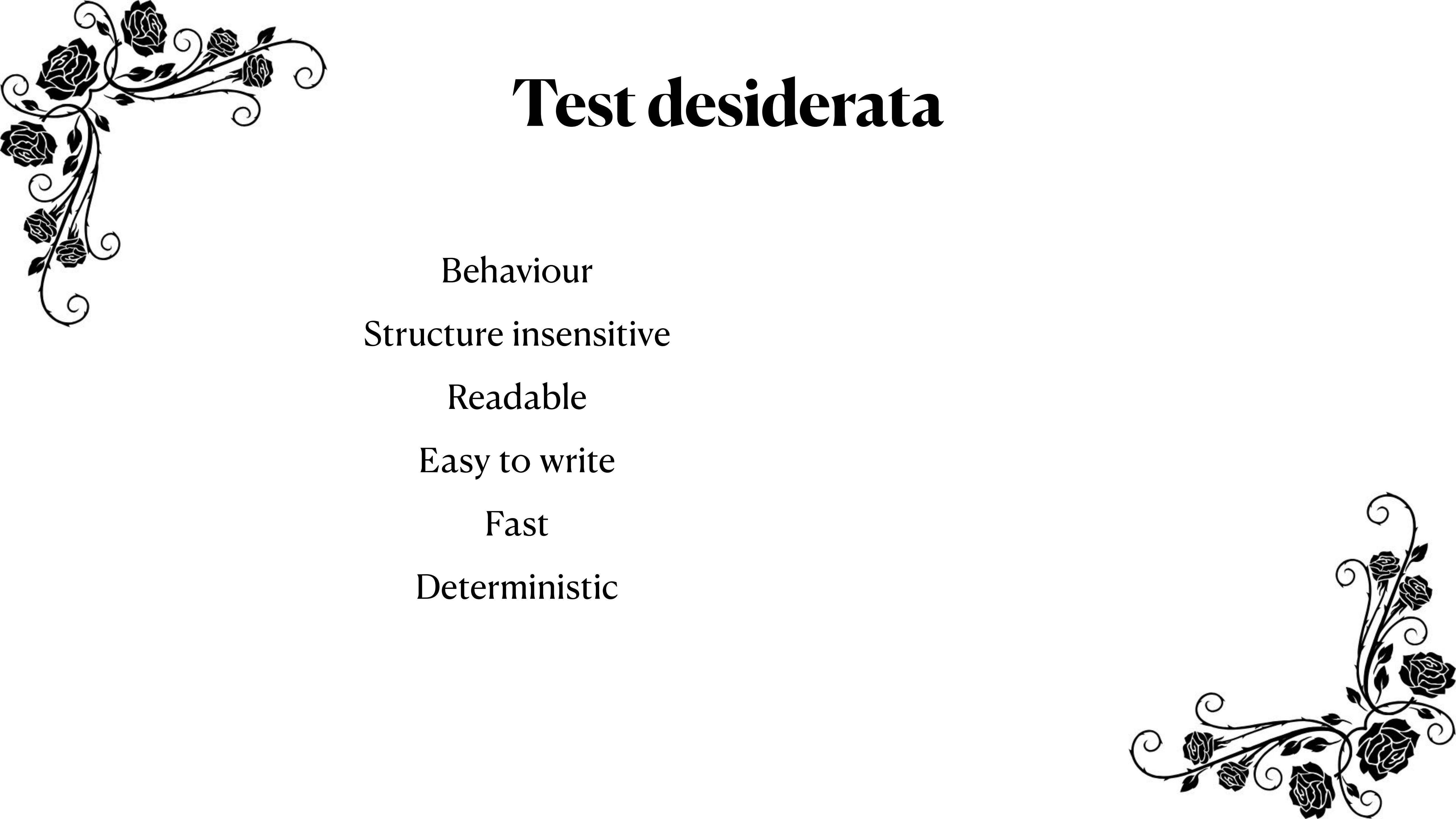
Structure insensitive

Readable

Easy to write

Fast





# Test desiderata

Behaviour

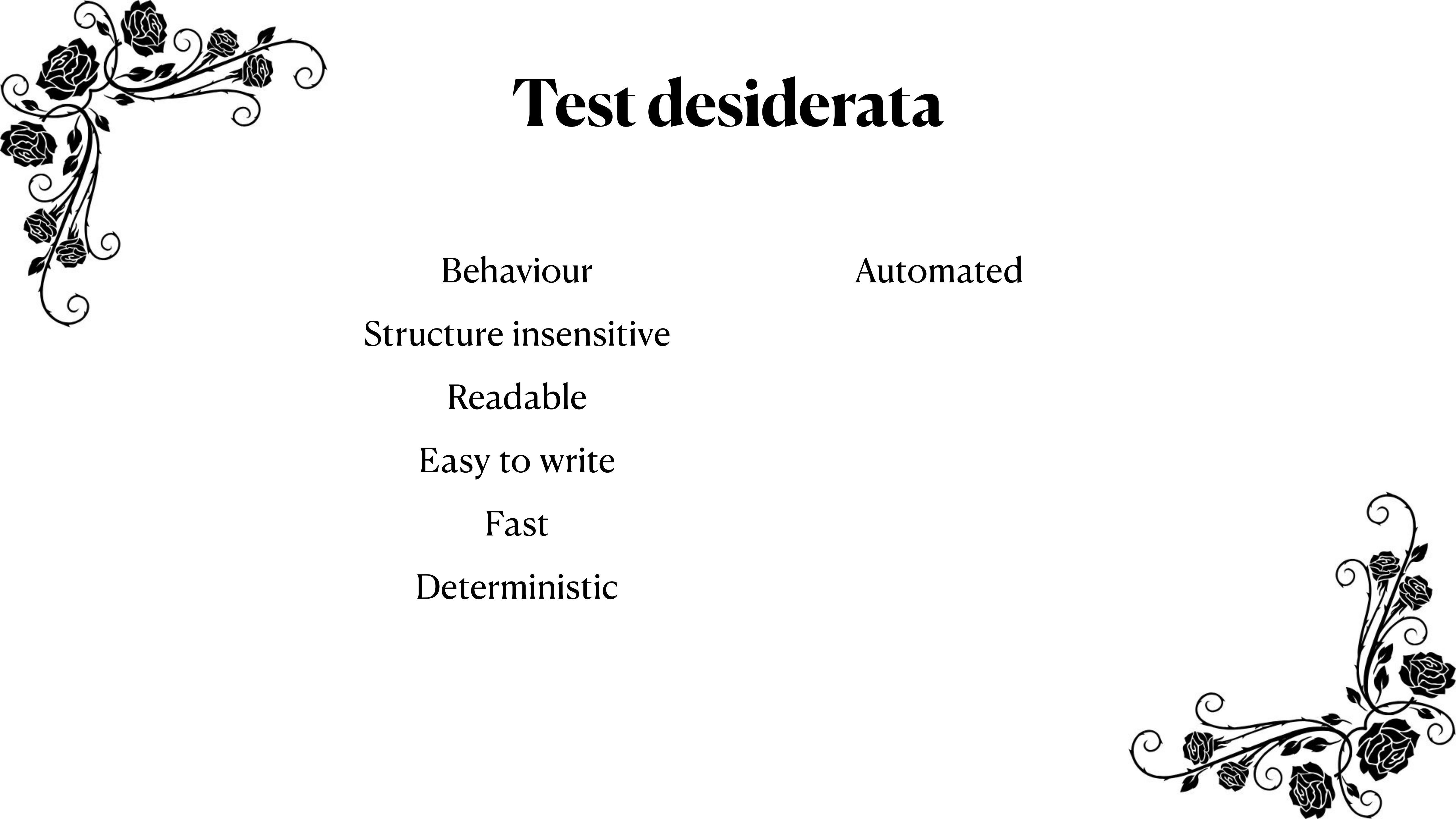
Structure insensitive

Readable

Easy to write

Fast

Deterministic



# Test desiderata

Behaviour

Automated

Structure insensitive

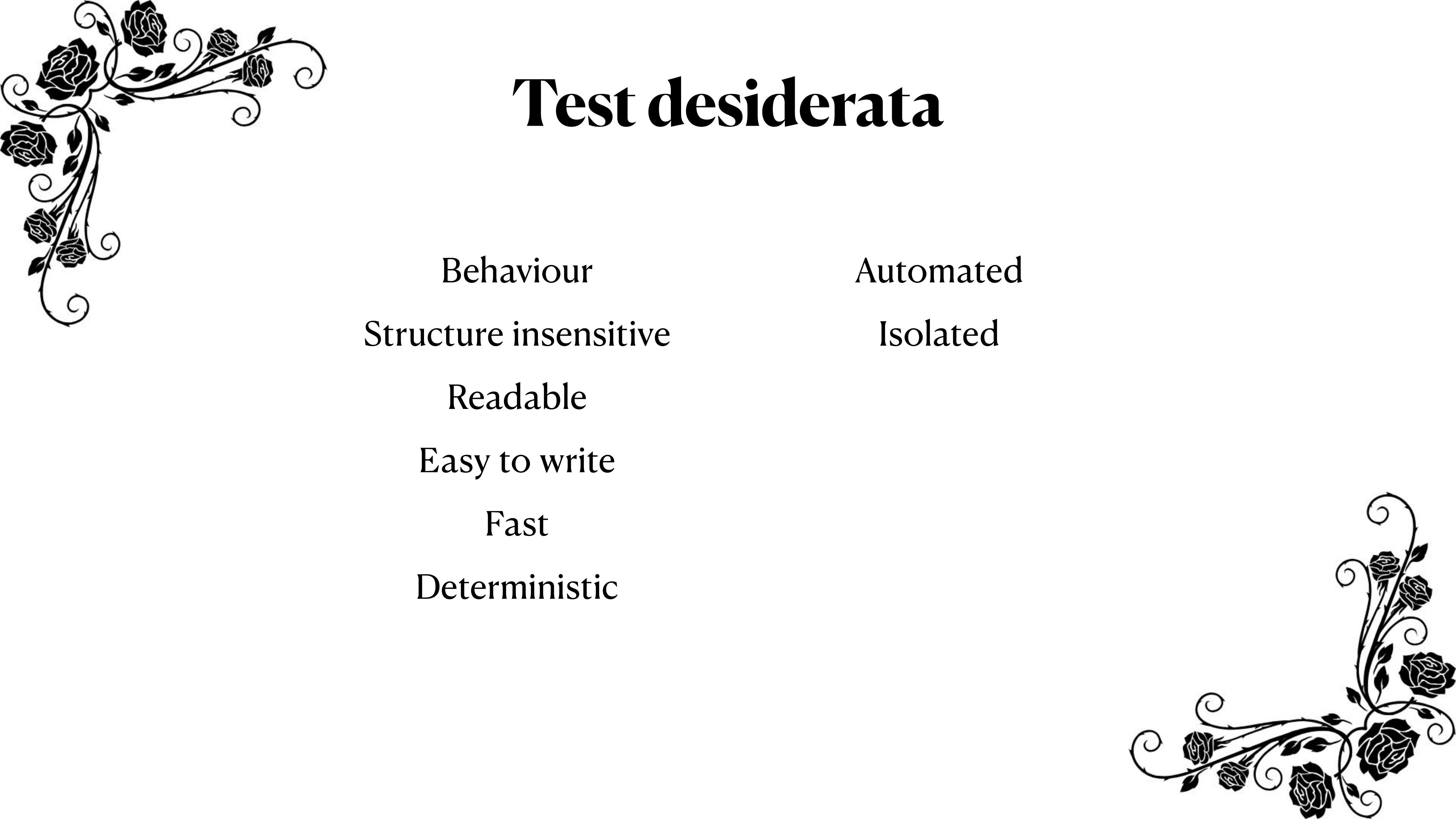
Readable

Easy to write

Fast

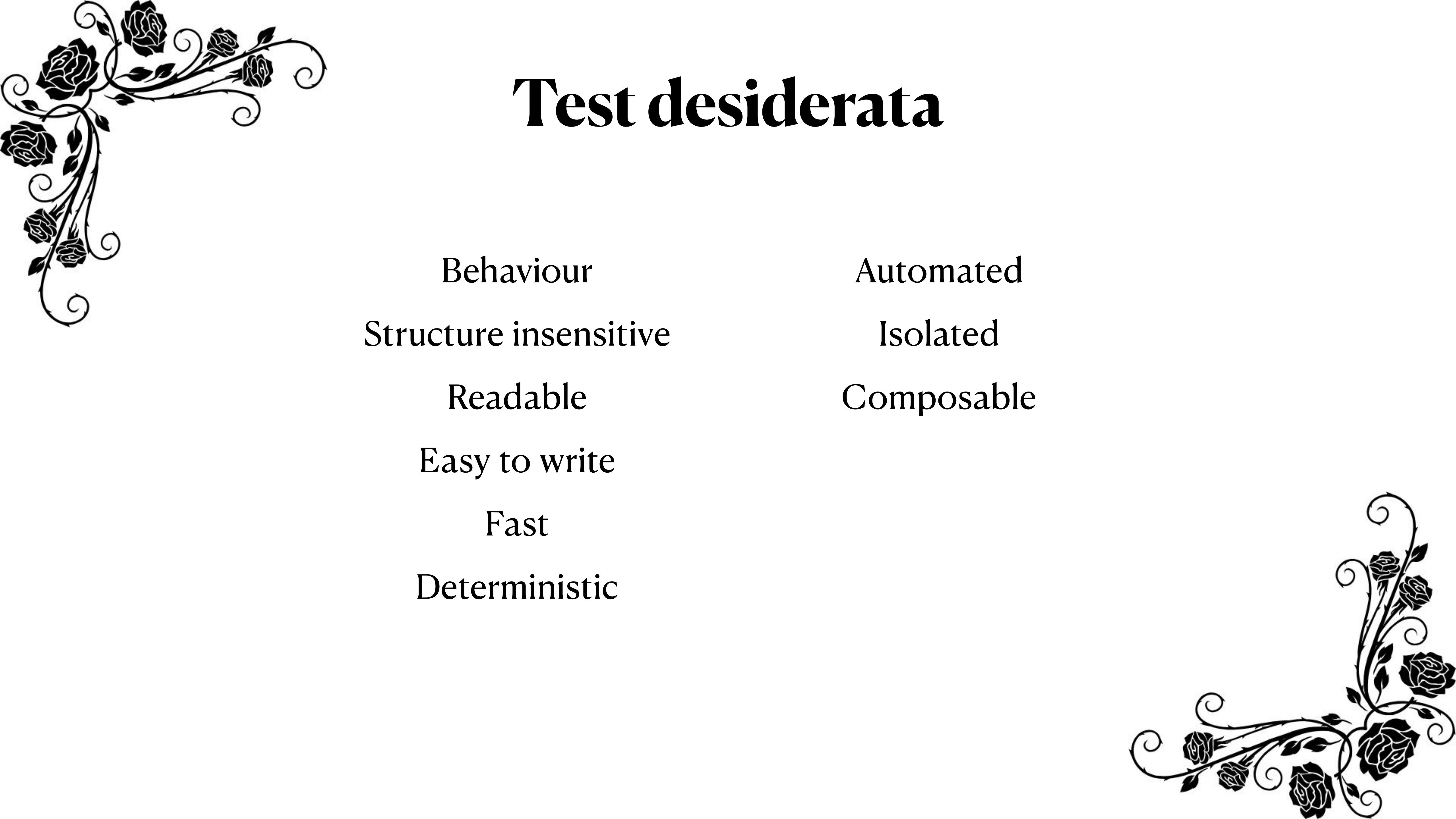
Deterministic





# Test desiderata

Behaviour	Automated
Structure insensitive	Isolated
Readable	
Easy to write	
Fast	
Deterministic	



# Test desiderata

Behaviour

Structure insensitive

Readable

Easy to write

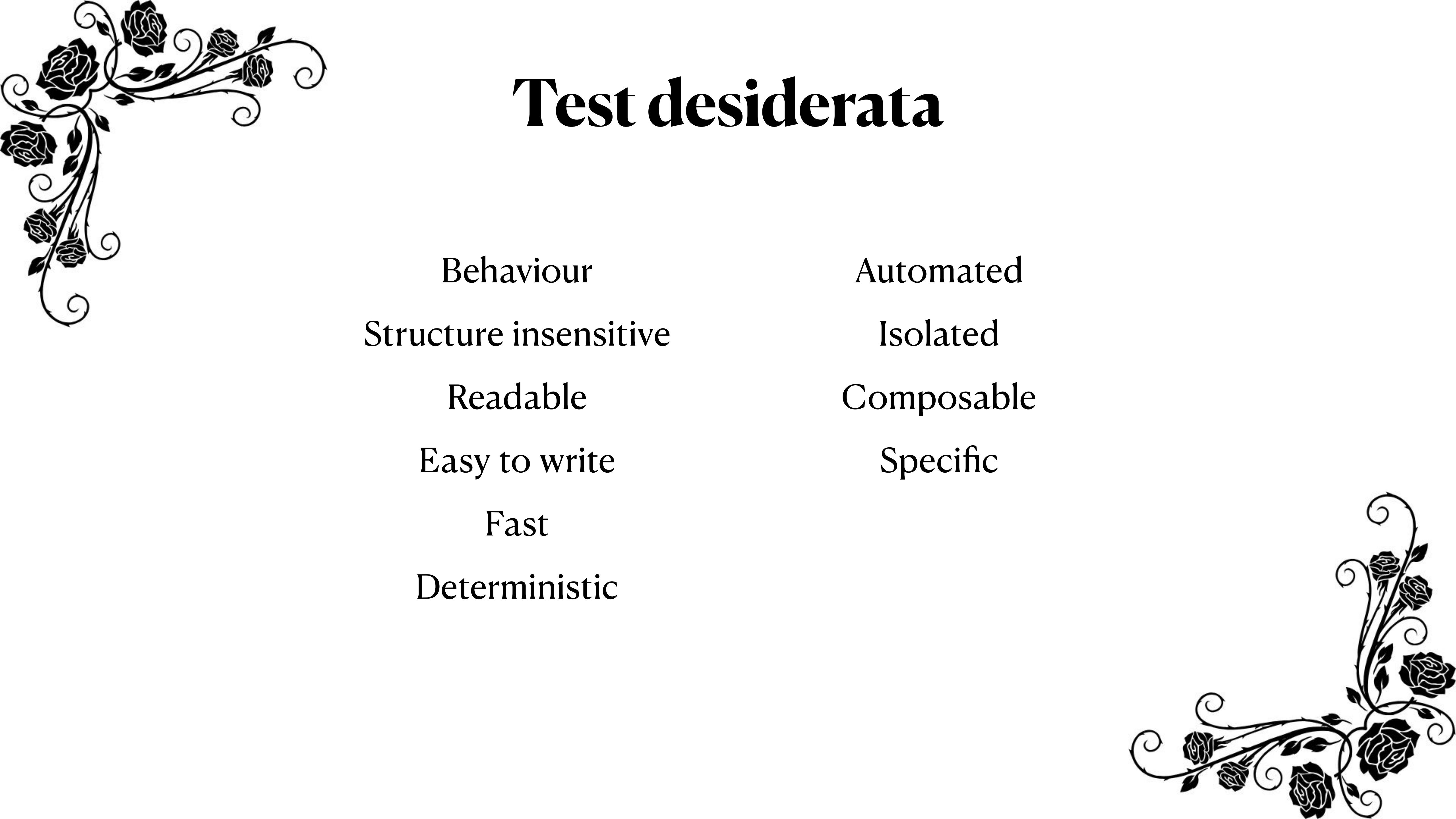
Fast

Deterministic

Automated

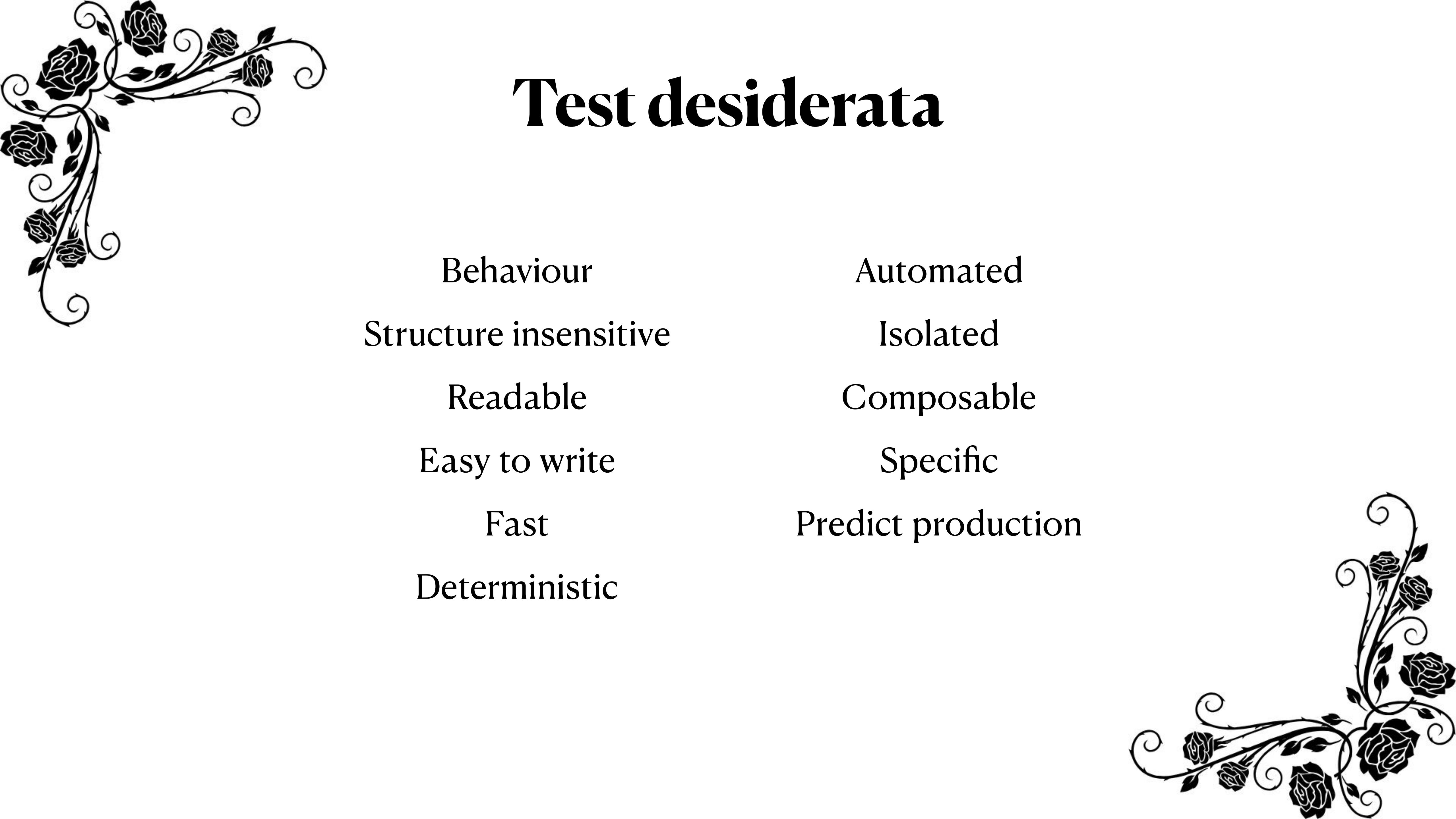
Isolated

Composable



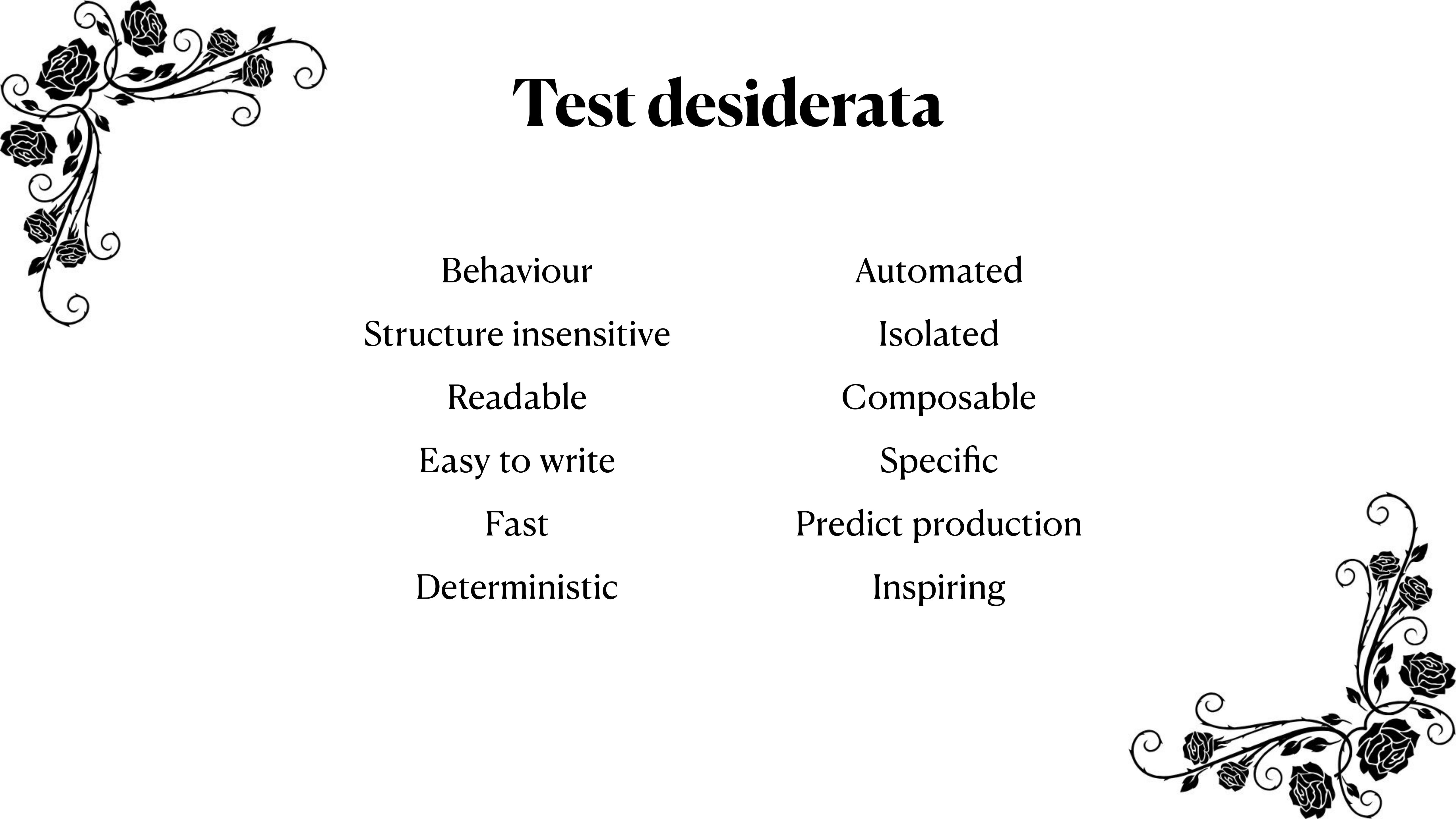
# Test desiderata

Behaviour	Automated
Structure insensitive	Isolated
Readable	Composable
Easy to write	Specific
Fast	
Deterministic	



# Test desiderata

Behaviour	Automated
Structure insensitive	Isolated
Readable	Composable
Easy to write	Specific
Fast	Predict production
Deterministic	



# Test desiderata

Behaviour	Automated
Structure insensitive	Isolated
Readable	Composable
Easy to write	Specific
Fast	Predict production
Deterministic	Inspiring

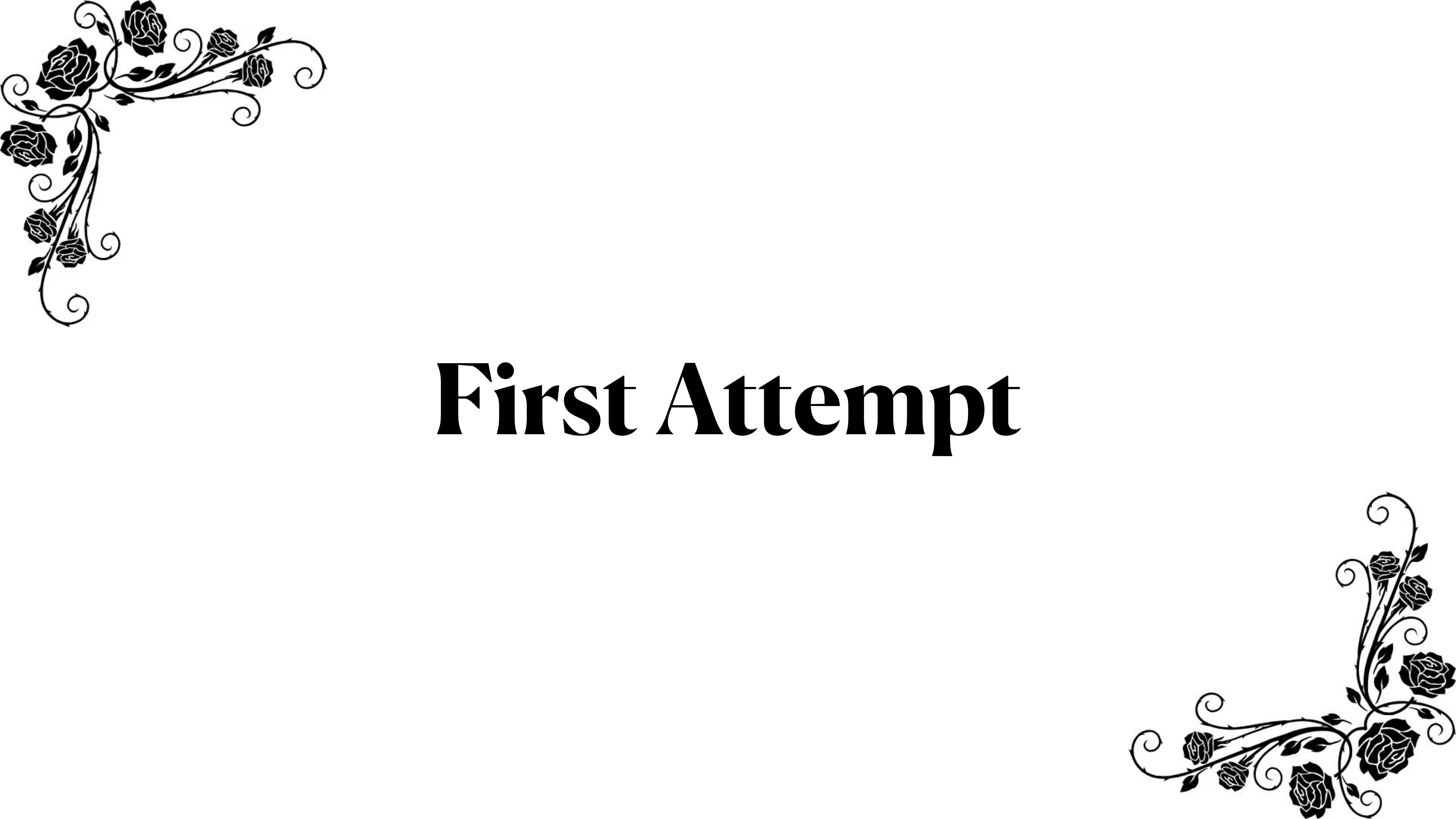
# The challenge



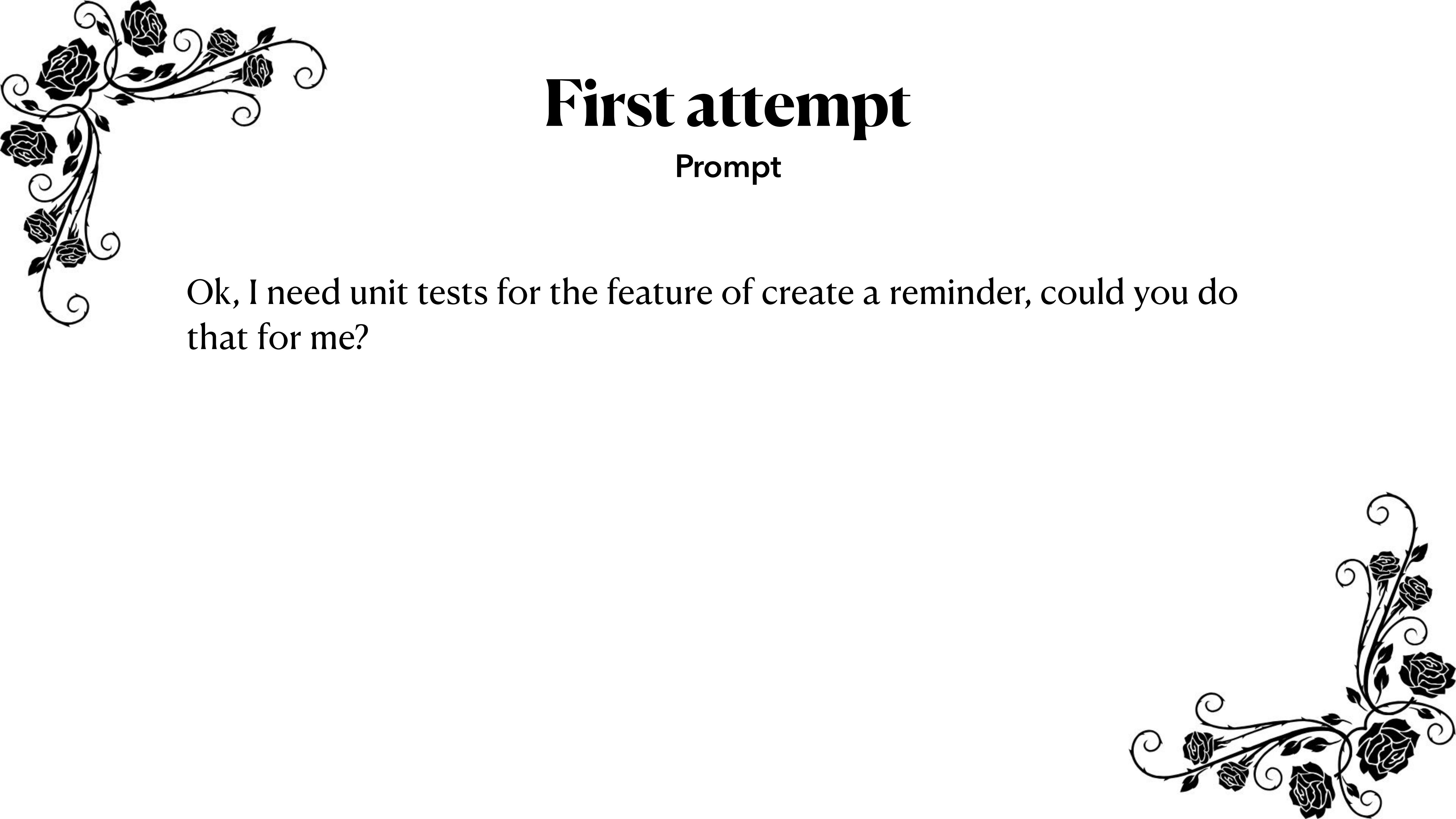
**<https://github.com/isamadrid90/modern-testing-tale-example>**

# Context

- New feature to create reminders for a vet, the details are in a file DESCRIPTION.md, this file was used as context for third AI prompt
- The feature is developed and I want to add test to that feature, so the code developed will be used as context for each AI prompt



# First Attempt



# **First attempt**

## **Prompt**

Ok, I need unit tests for the feature of create a reminder, could you do that for me?



```
● ○ ●

@Test
fun `createReminder should return complete response correctly`() {
    val reminderService: ReminderService = mockk(relaxUnitFun = true)
    val createReminderController: CreateReminderController = CreateReminderController(reminderService)

    val request = ReminderRequest(
        vetEmail = "vet@example.com",
        timeOffset = TimeOffset(amount = 1, unit = "DAYS"),
        weekDay = "Monday",
        petName = "Buddy",
        petBirthday = "2020-01-01",
        ownerEmail = "owner@example.com",
        reminderType = "FOLLOW_UP",
        message = "Vet appointment for Buddy"
    )

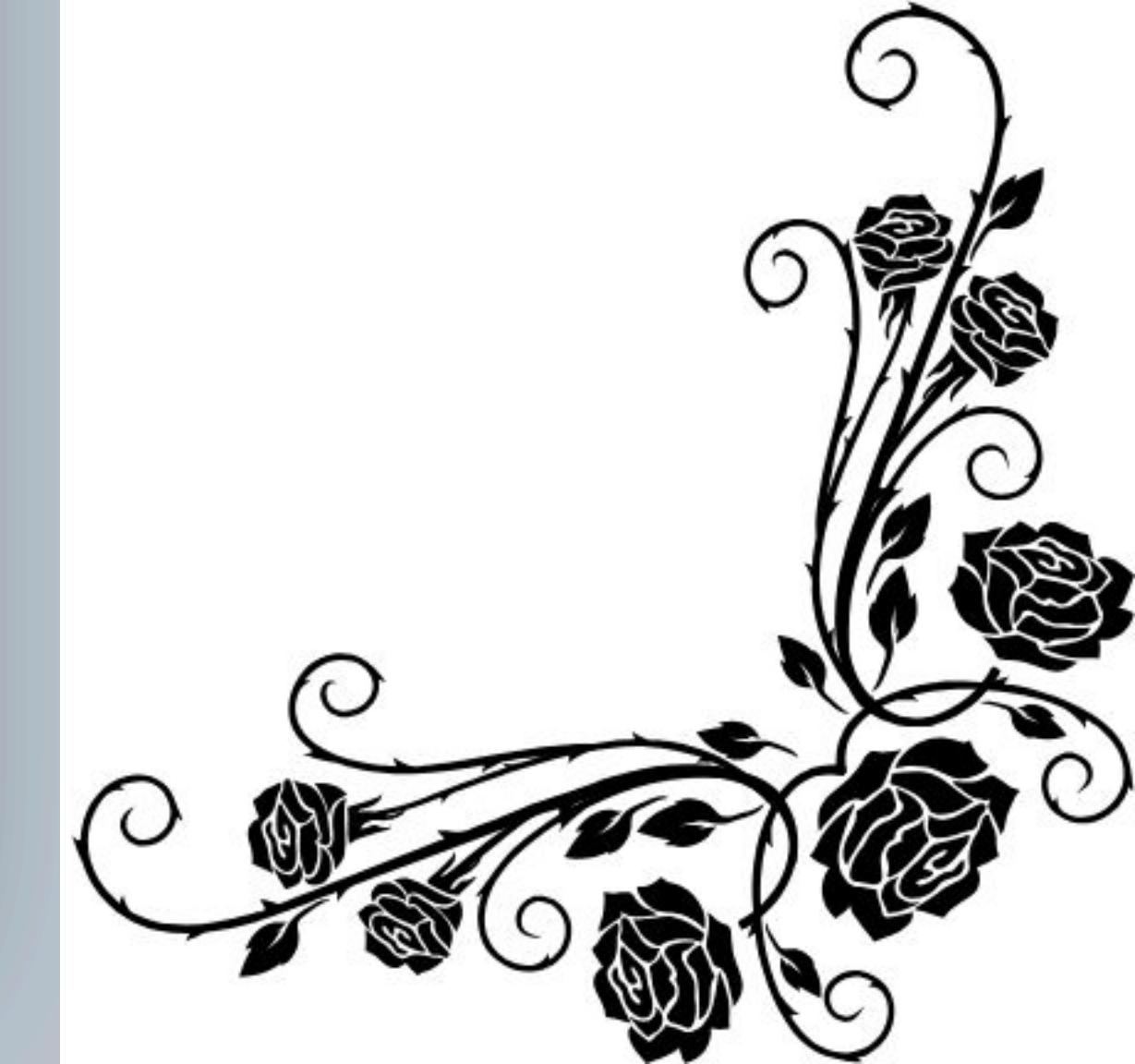
    val reminder = Reminder(
        id = "1",
        vetEmail = "vet@example.com",
        scheduledDate = LocalDateTime.of(2023, 10, 14, 10, 0),
        petName = "Buddy",
        petBirthday = LocalDate.parse("2020-01-01"),
        ownerEmail = "owner@example.com",
        reminderType = ReminderType.FOLLOW_UP,
        message = "Vet appointment for Buddy",
        createdAt = LocalDateTime.now()
    )

    val expectedResponse = with(reminder) {
        ReminderResponse(
            vetEmail = vetEmail,
            reminderDateTime = scheduledDate.format(DateTimeFormatter.ISO_LOCAL_DATE),
            pet = Pet(name = petName, birthday = petBirthday.toString(), ownerEmail = ownerEmail),
            reminderType = reminderType.type,
            message = message
        )
    }

    every { reminderService.createReminder(any()) } returns reminder

    val response: ResponseEntity<Any> = createReminderController.createReminder(request)

    Assertions.assertEquals(HttpStatus.OK, response.statusCode)
    Assertions.assertEquals(expectedResponse, response.body)
    verify { reminderService.createReminder(request) }
}
```



```
    @Test
    fun `createReminder should handle invalid date format`() {
        val reminderService = mockk<ReminderService>()
        val controller = CreateReminderController(reminderService)
        val request = ReminderRequest(
            vetEmail = "vet@example.com",
            timeOffset = TimeOffset(amount = 1, unit = "DAYS"),
            weekDay = "Monday",
            petName = "Buddy",
            petBirthday = "invalid-date-format",
            ownerEmail = "owner@example.com",
            reminderType = "FOLLOW_UP",
            message = "Vet appointment for Buddy"
        )

        every { reminderService.createReminder(any()) } throws InvalidPetBirthdateException("Invalid pet
birthdate format")

        val response: ResponseEntity<Any> = controller.createReminder(request)

        Assertions.assertEquals(HttpStatus.BAD_REQUEST, response.statusCode)
    }
```



```
@Test
```

```
fun `createReminder should handle missing required field`() {
    val reminderService: ReminderService = mockk(relaxUnitFun = true)
    val createReminderController: CreateReminderController =
        CreateReminderController(reminderService)

    val request = ReminderRequest(
        vetEmail = "vet@example.com",
        timeOffset = TimeOffset(amount = 1, unit = "DAYS"),
        weekDay = "Monday",
        petName = "Buddy",
        petBirthday = "2020-01-01",
        ownerEmail = "owner@example.com",
        reminderType = "FOLLOW_UP",
        message = "" // Missing required field
    )
    every { reminderService.createReminder(any()) } throws IllegalArgumentException("Missing
required field")

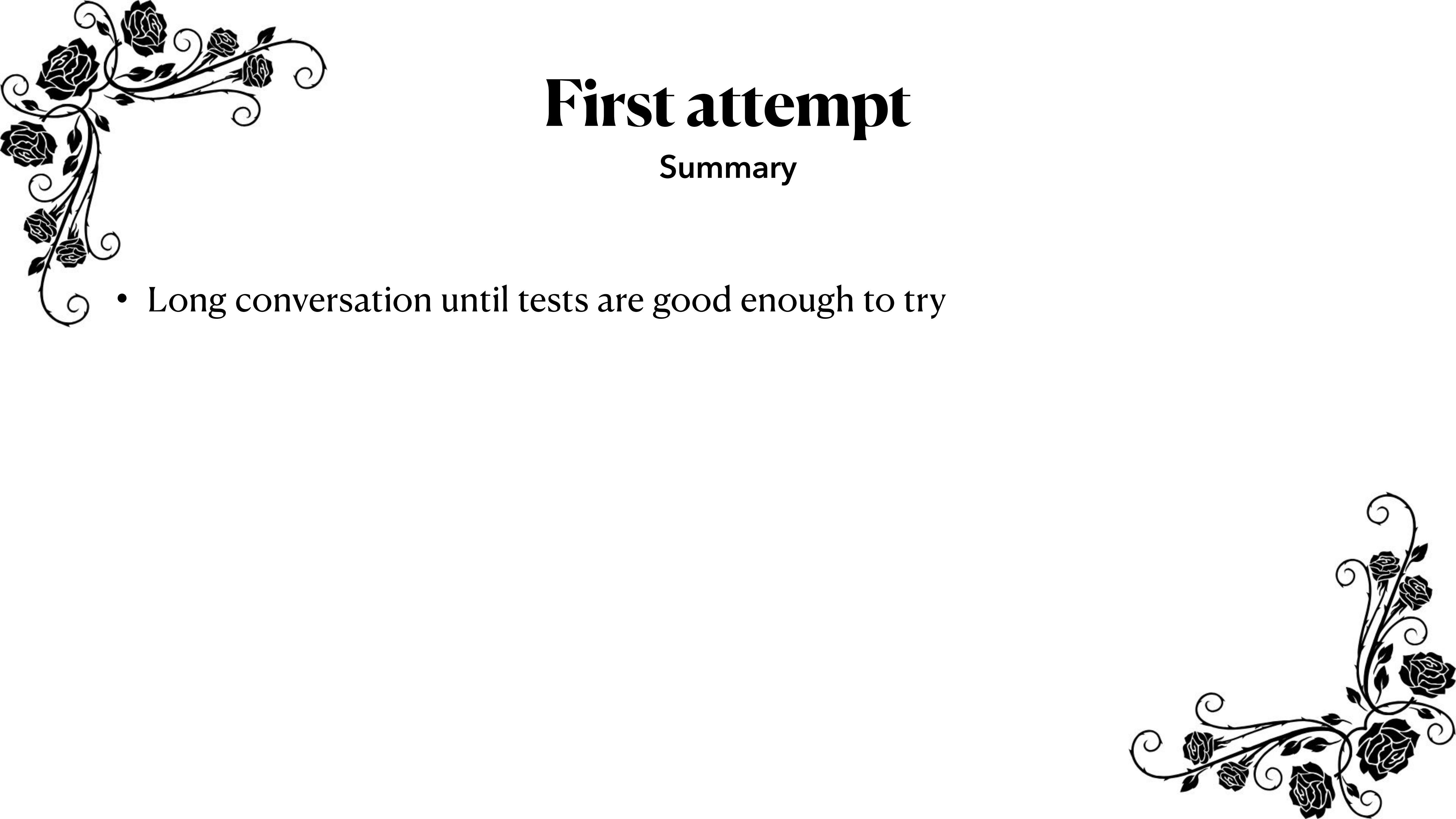
    val exception = Assertions.assertThrows(IllegalArgumentException::class.java) {
        createReminderController.createReminder(request)
    }

    Assertions.assertEquals("Missing required field", exception.message)
}
```



# **First attempt**

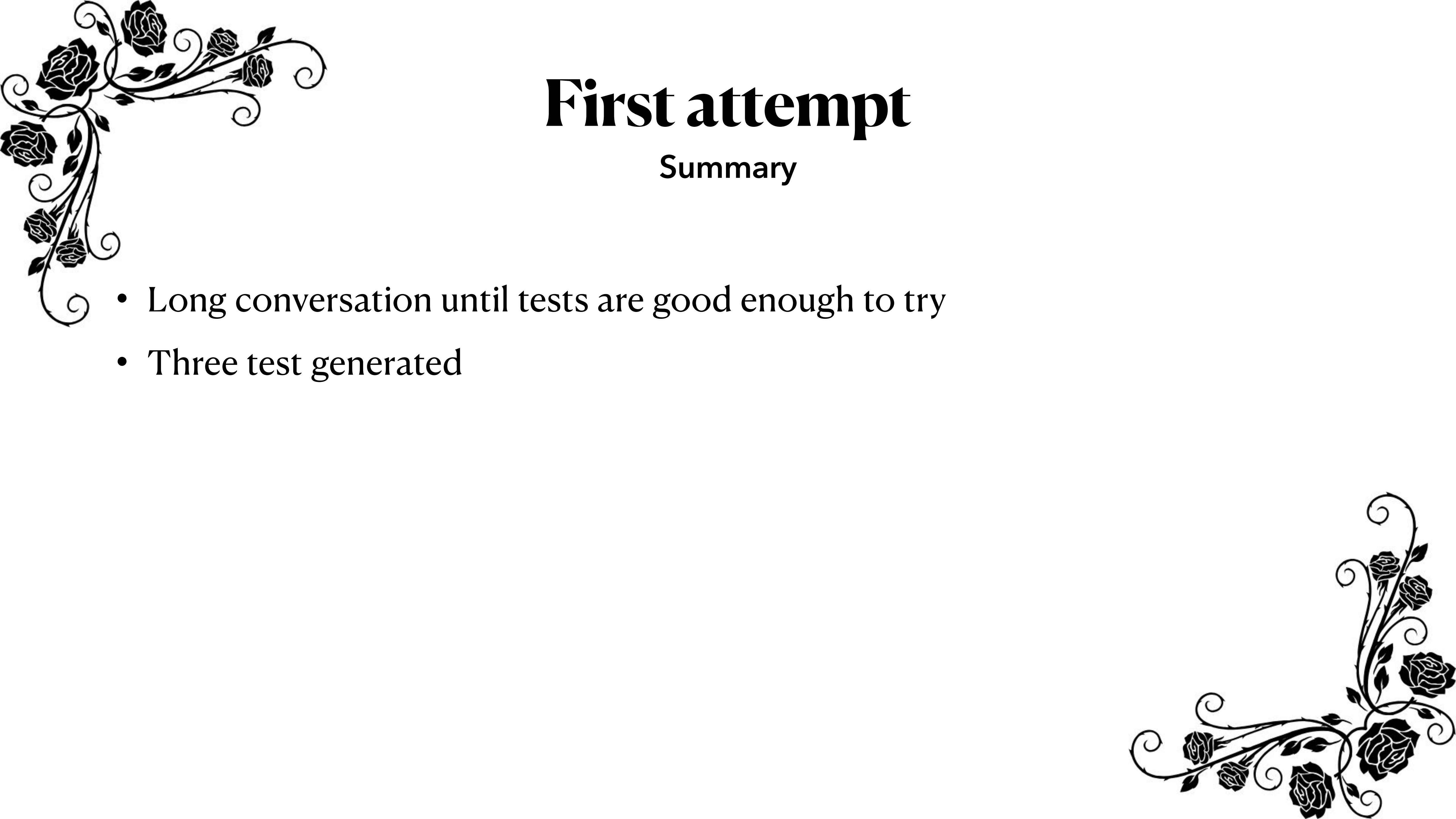
## **Summary**



# **First attempt**

## **Summary**

- Long conversation until tests are good enough to try



# First attempt

## Summary

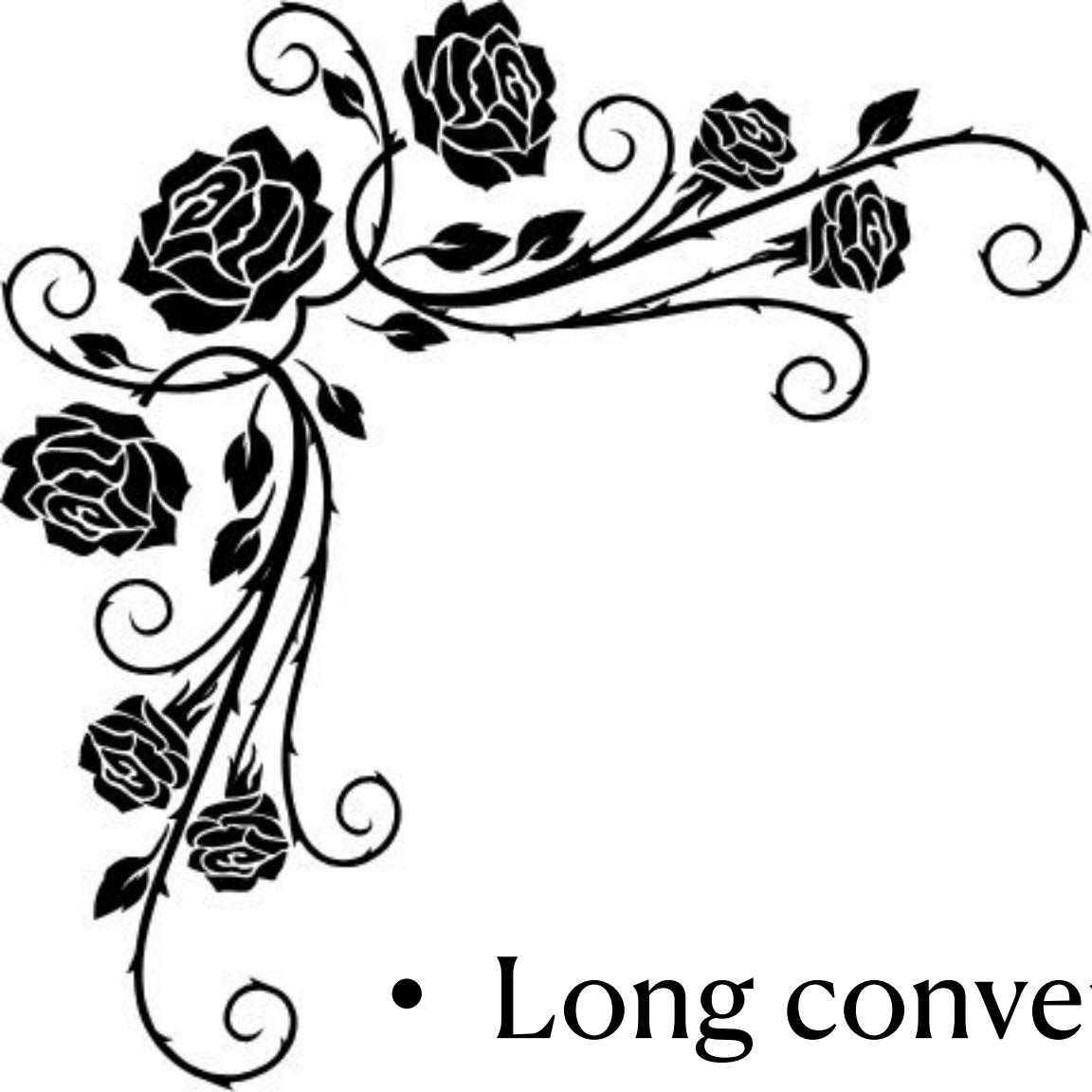
- Long conversation until tests are good enough to try
- Three test generated



# First attempt

## Summary

- Long conversation until tests are good enough to try
  - Three test generated
  - Test didn't pass 
- 



# First attempt

## Summary

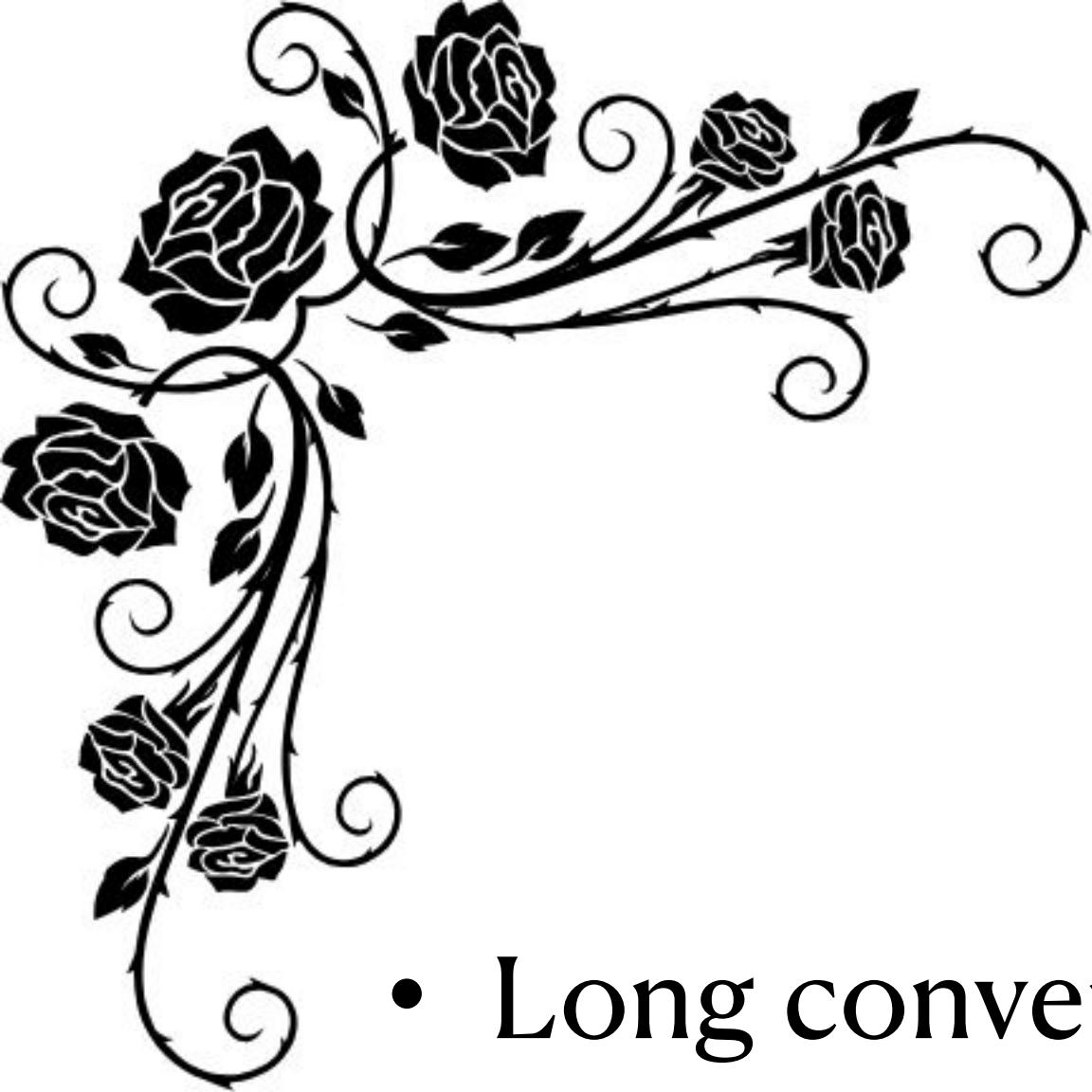
- Long conversation until tests are good enough to try
  - Three test generated
  - Test didn't pass **✗**
  - Tests are easy to read
- 



# First attempt

## Summary

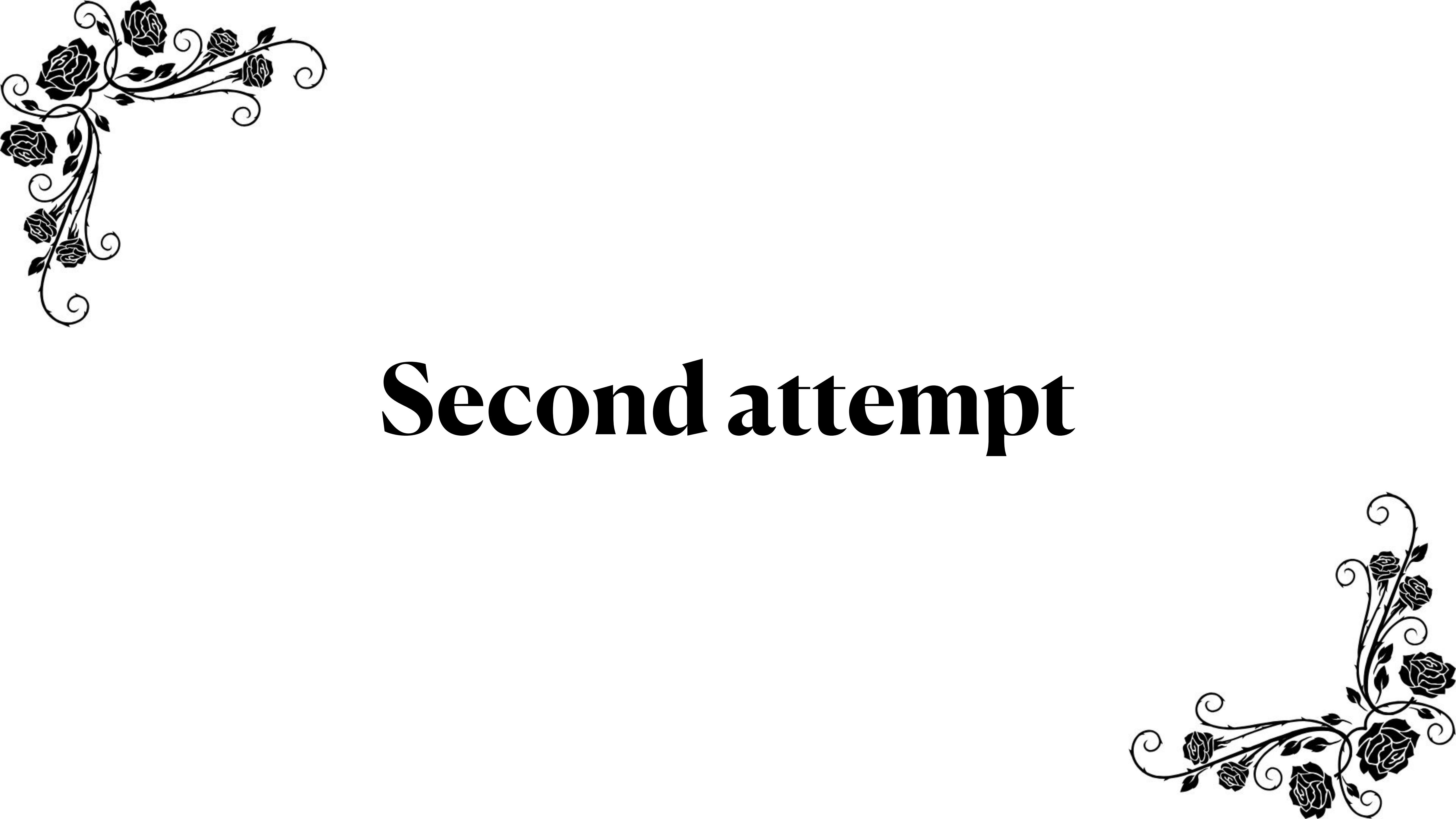
- Long conversation until tests are good enough to try
  - Three test generated
  - Test didn't pass **✗**
  - Tests are easy to read
  - Tests not deterministic (using Localdatetime.now)
- 



# First attempt

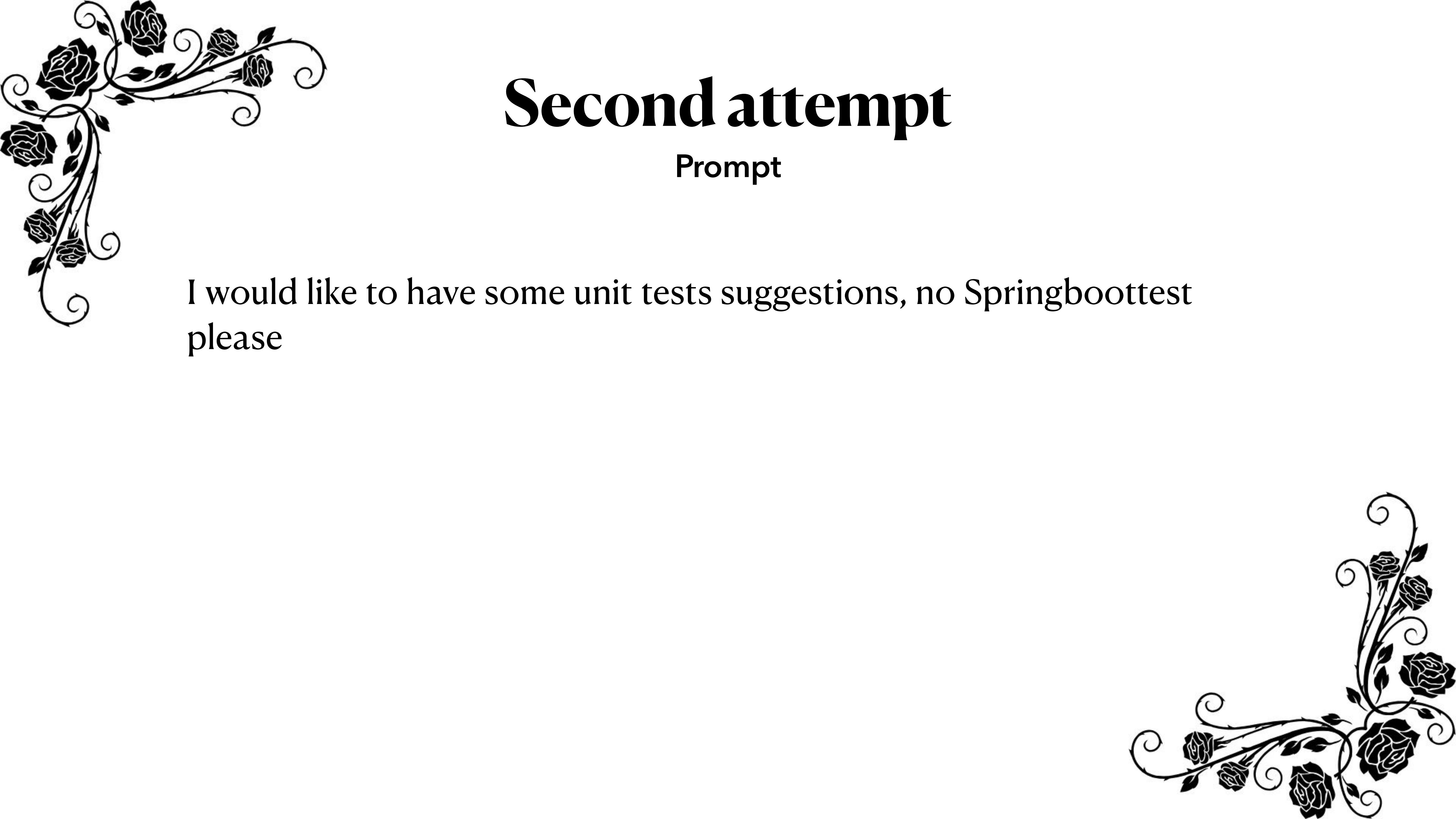
## Summary

- Long conversation until tests are good enough to try
  - Three test generated
  - Test didn't pass **X**
  - Tests are easy to read
  - Tests not deterministic (using Localdatetime.now)
  - Test for failures (not checking response code)
- 



A decorative border consisting of black roses and swirling vines in the corners of the page.

# Second attempt



# **Second attempt**

**Prompt**

I would like to have some unit tests suggestions, no Springboottest  
please



```
● ● ●

@ExtendWith(MockKExtension::class)
class CreateReminderControllerTest {

    @MockK
    private lateinit var reminderService: ReminderService

    private lateinit var createReminderController: CreateReminderController

    @BeforeEach
    fun setUp() {
        createReminderController = CreateReminderController(reminderService)
    }

    @Test
    fun `should create reminder and return created status`() {
        val request = ReminderRequest(
            vetEmail = "vet@example.com",
            timeOffset = TimeOffset(amount = 2, unit = "days"),
            weekDay = "Monday",
            petName = "Buddy",
            petBirthday = "2022-08-20",
            ownerEmail = "owner@example.com",
            reminderType = "Vaccination",
            message = "Time for your pet's vaccination!"
        )

        val scheduledDate = LocalDateTime.now().plusDays(2)
        val reminder = Reminder(
            id = "1",
            vetEmail = "vet@example.com",
            scheduledDate = scheduledDate,
            petName = "Buddy",
            petBirthday = LocalDate.parse("2022-08-20"),
            ownerEmail = "owner@example.com",
            reminderType = ReminderType.CHECK_UP,
            message = "Time for your pet's vaccination!",
            createdAt = LocalDateTime.now()
        )

        val objectMapper = ObjectMapper()

        // Mock behavior
        every { reminderService.createReminder(any<ReminderRequest>()) } returns reminder

        val response: ResponseEntity<Any> = createReminderController.createReminder(request)

        assertEquals(HttpStatus.CREATED, response.statusCode)
        assertEquals(objectMapper.writeValueAsString(reminder),
                    objectMapper.writeValueAsString(response.body))
    }
}
```





```
● ● ●

@ExtendWith(MockKExtension::class)
class ReminderServiceTest {

    @MockK
    private lateinit var reminderRepository: ReminderRepository

    @MockK
    private lateinit var reminderDateTimeCalculator: ReminderDateTimeCalculator

    private lateinit var reminderService: ReminderService

    @BeforeEach
    fun setUp() {
        reminderService = ReminderService(reminderRepository, reminderDateTimeCalculator)
    }

    @Test
    fun `should create reminder successfully`() {
        val request = ReminderRequest(
            vetEmail = "vet@example.com",
            timeOffset = TimeOffset(amount = 2, unit = "days"),
            weekDay = "Monday",
            petName = "Buddy",
            petBirthday = "2022-08-20",
            ownerEmail = "owner@example.com",
            reminderType = "Vaccination",
            message = "Time for your pet's vaccination!"
        )

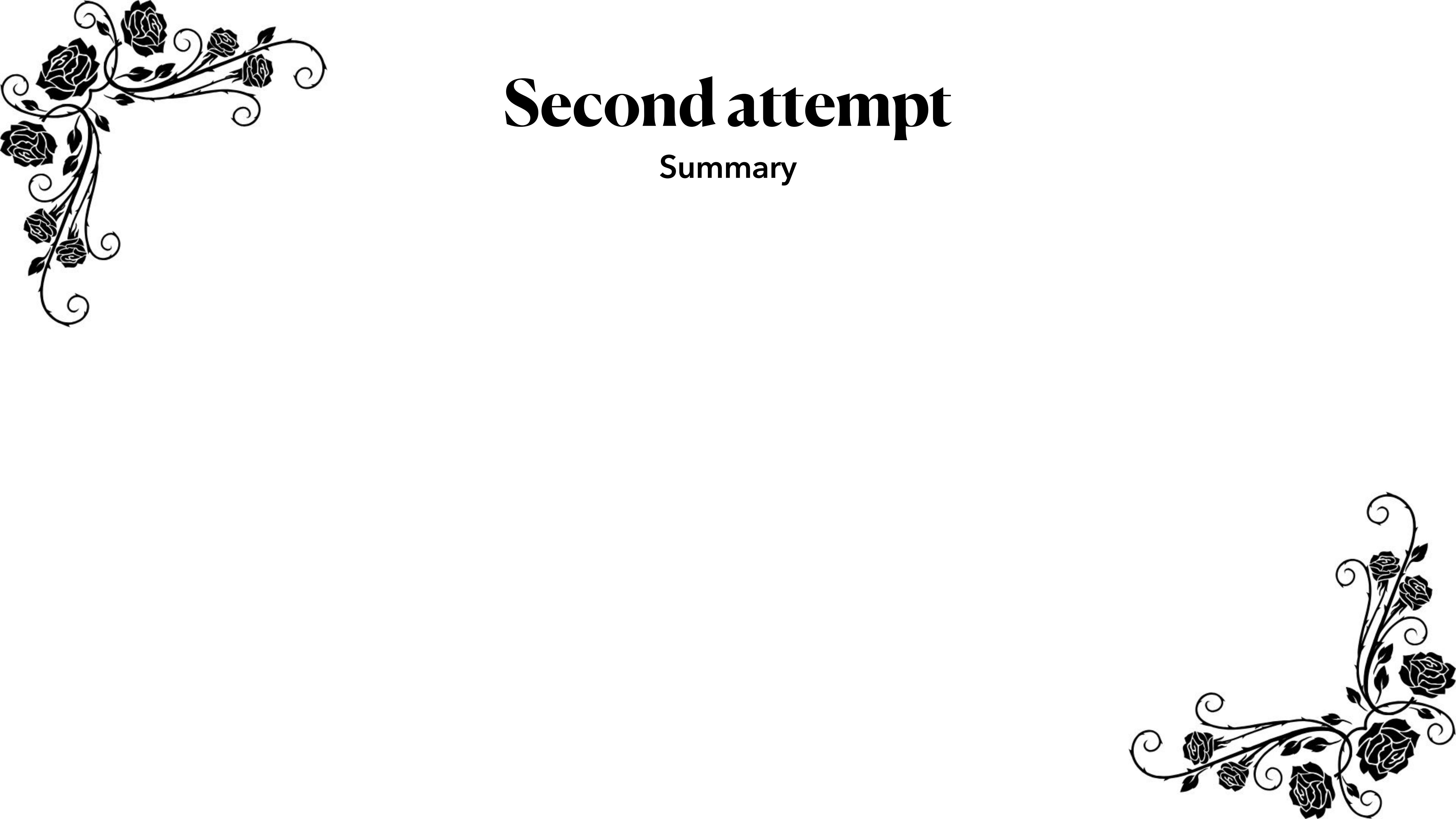
        val scheduledDate = LocalDateTime.now().plusDays(2)
        val reminder = Reminder(
            id = "1",
            vetEmail = "vet@example.com",
            scheduledDate = scheduledDate,
            petName = "Buddy",
            petBirthday = LocalDate.parse("2022-08-20"),
            ownerEmail = "owner@example.com",
            reminderType = ReminderType.VACCINATION,
            message = "Time for your pet's vaccination!",
            createdAt = LocalDateTime.now()
        )

        // Mock behavior
        every { reminderDateTimeCalculator.calculate(any()) } returns scheduledDate
        every { reminderRepository.save(any<Reminder>()) } returns reminder

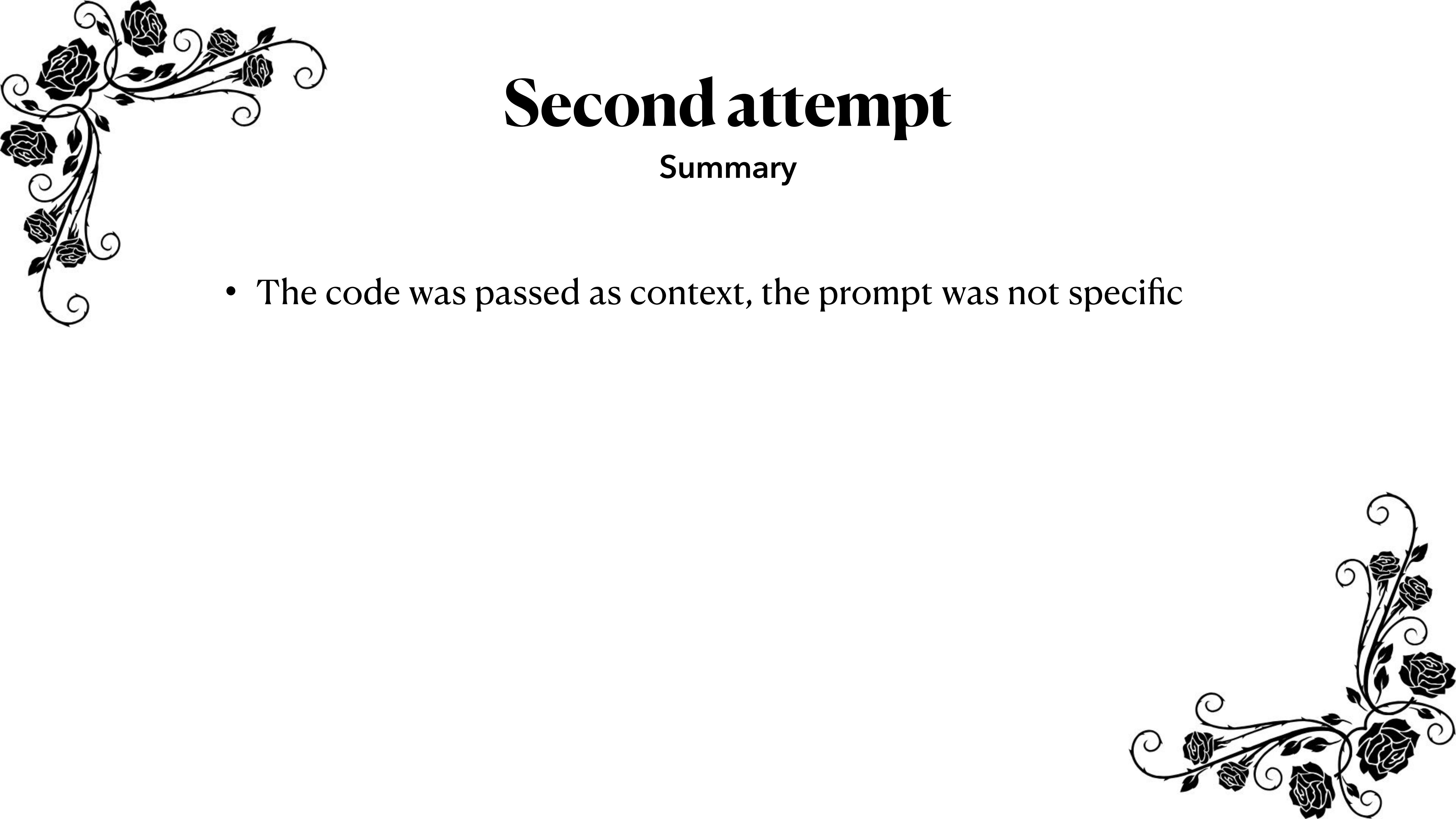
        val result = reminderService.createReminder(request)

        assertNotNull(result)
        assertEquals(reminder, result)
        verify { reminderRepository.save(any<Reminder>()) }
    }
}
```





# **Second attempt****Summary**



# **Second attempt**

## **Summary**

- The code was passed as context, the prompt was not specific



# Second attempt

## Summary

- The code was passed as context, the prompt was not specific
  - Two test generated
- 



# Second attempt

## Summary

- The code was passed as context, the prompt was not specific
  - Two test generated
  - Test didn't pass **✗**
- 



# Second attempt

## Summary

- The code was passed as context, the prompt was not specific
  - Two test generated
  - Test didn't pass **✗**
  - Tests are easy to read
- 



# Second attempt

## Summary

- The code was passed as context, the prompt was not specific
  - Two test generated
  - Test didn't pass **✗**
  - Tests are easy to read
  - Tests not deterministic (using Localdatetime.now)
- 



# Second attempt

## Summary

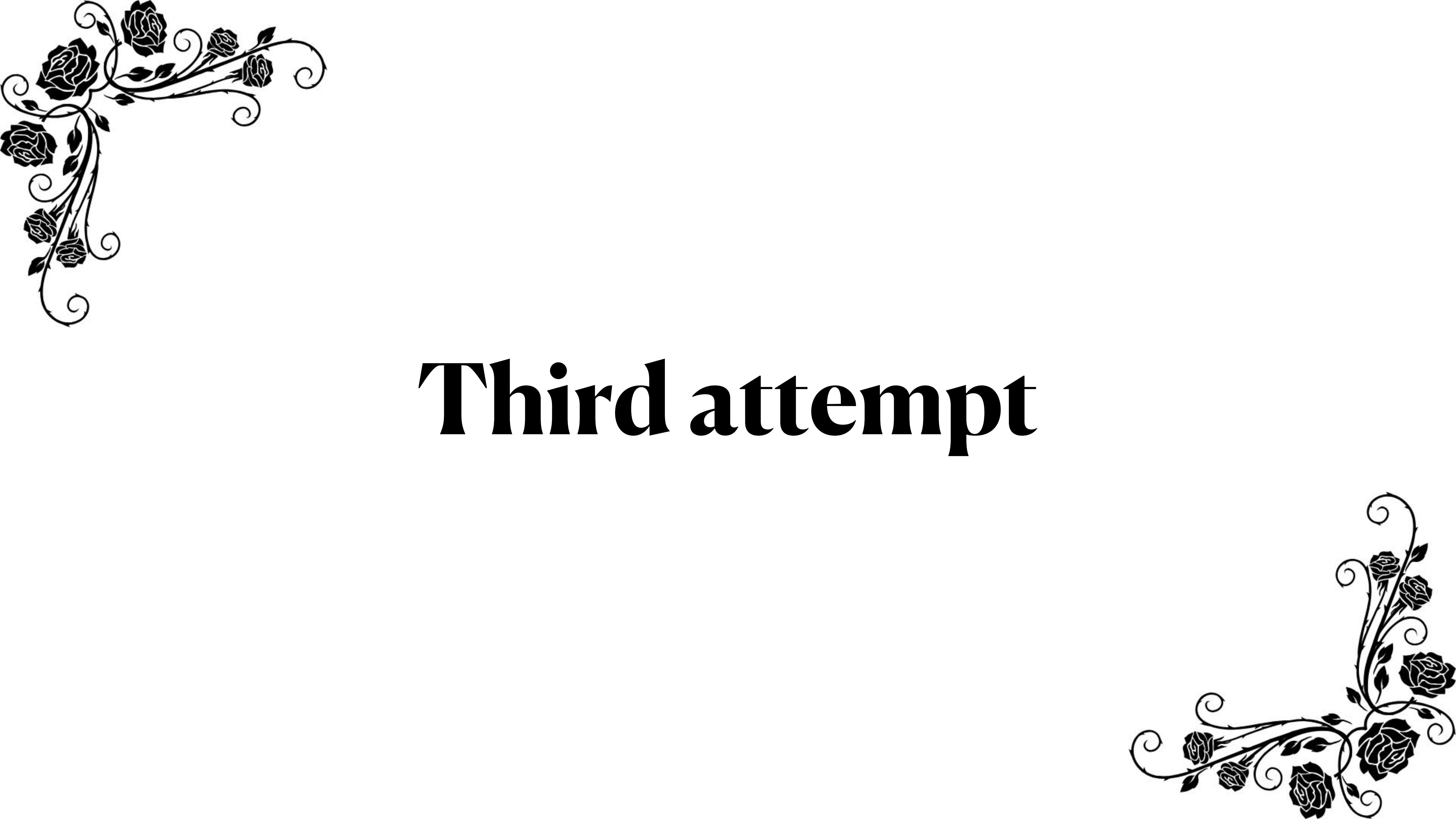
- The code was passed as context, the prompt was not specific
  - Two test generated
  - Test didn't pass **✗**
  - Tests are easy to read
  - Tests not deterministic (using Localdatetime.now)
  - No test for failures
- 



# Second attempt

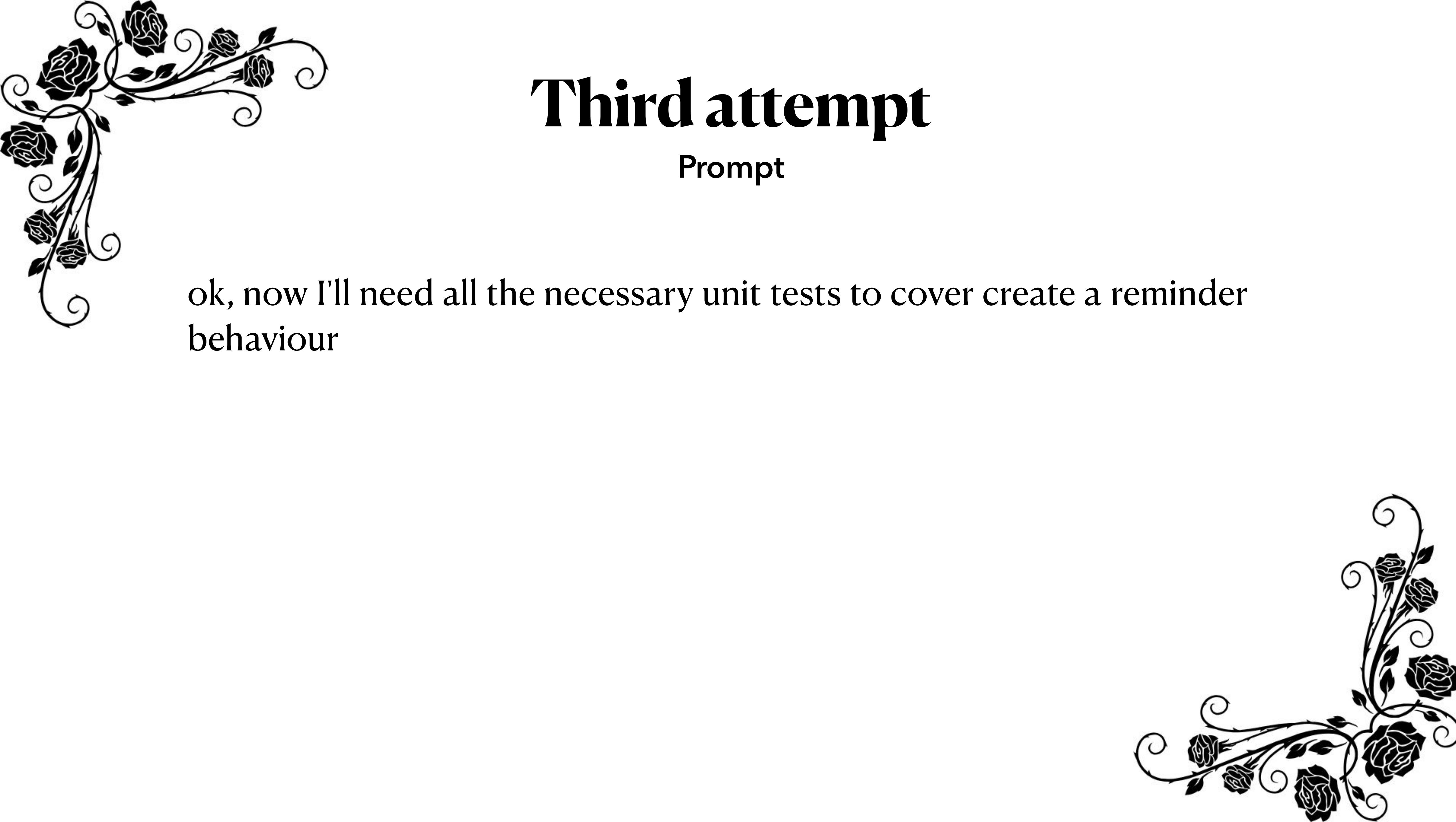
## Summary

- The code was passed as context, the prompt was not specific
  - Two test generated
  - Test didn't pass **✗**
  - Tests are easy to read
  - Tests not deterministic (using Localdatetime.now)
  - No test for failures
  - Missing tests for critical collaborators
- 



A decorative border consisting of black roses and swirling vines in the corners of the image.

# Third attempt



# Third attempt

## Prompt

ok, now I'll need all the necessary unit tests to cover create a reminder behaviour



```
● ● ●
```

```
class CreateReminderControllerTest {

    private val reminderService: ReminderService = mockk()
    private val createReminderController = CreateReminderController(reminderService)

    @Test
    fun `should create a reminder successfully`() {
        val request = CreateReminderRequest(
            vetEmail = "vet@example.com",
            timeToAdd = "30",
            weekDay = "Monday",
            petName = "Buddy",
            birthday = "2020-01-01",
            ownerEmail = "owner@example.com",
            reminderType = "Check-up",
            message = "Time for the annual check-up!"
        )

        every { reminderService.createReminder(any()) } returns ReminderResponse(
            success = true,
            message = "Reminder created successfully"
        )

        val response: ResponseEntity<ReminderResponse> =
        createReminderController.createReminder(request)

        assertEquals(HttpStatus.OK, response.statusCode)
        assertEquals("Reminder created successfully", response.body?.message)

        verify(exactly = 1) { reminderService.createReminder(any()) }
    }

    @Test
    fun `should handle invalid date format exception`() {
        val request = CreateReminderRequest(
            vetEmail = "vet@example.com",
            timeToAdd = "invalid",
            weekDay = "Monday",
            petName = "Buddy",
            birthday = "2020-01-01",
            ownerEmail = "owner@example.com",
            reminderType = "Check-up",
            message = "Time for the annual check-up!"
        )

        every { reminderService.createReminder(any()) } throws IllegalArgumentException("Invalid date
format")

        val exception: IllegalArgumentException = assertThrows(IllegalArgumentException::class.java) {
            createReminderController.createReminder(request)
        }

        assertEquals("Invalid date format", exception.message)

        verify(exactly = 1) { reminderService.createReminder(any()) }
    }
}
```





```
class ReminderDateTimeCalculatorTest {

    private val reminderDateTimeCalculator = ReminderDateTimeCalculator()

    @Test
    fun `should calculate future date matching given weekday`() {
        val currentDate = LocalDate.of(2023, 10, 1)
        val timeToAdd = 30
        val weekDay = DayOfWeek.MONDAY

        val calculatedDate = reminderDateTimeCalculator.calculateFutureDate(currentDate, timeToAdd,
weekDay)

        assertEquals(DayOfWeek.MONDAY, calculatedDate.dayOfWeek)
        assertTrue(calculatedDate.isAfter(currentDate))
    }
}
```



```
class ReminderServiceTest {

    private val reminderDateTimeCalculator: ReminderDateTimeCalculator = mockk()
    private val reminderService = ReminderService(reminderDateTimeCalculator)

    @Test
    fun `should calculate correct reminder date`() {
        val vetEmail = "vet@example.com"
        val timeToAdd = "30"
        val weekDay = "Monday"
        val petName = "Buddy"
        val birthday = "2020-01-01"
        val ownerEmail = "owner@example.com"
        val reminderType = "Check-up"
        val message = "Time for the annual check-up!"

        every { reminderDateTimeCalculator.calculateFutureDate(any(), any(), any()) } returns
LocalDate.of(2023, 11, 1)

        val reminder = reminderService.createReminder(CreateReminderRequest(
            vetEmail, timeToAdd, weekDay, petName, birthday, ownerEmail, reminderType, message
        ))

        Assertions.assertNotNull(reminder)
        Assertions.assertEquals("Buddy", reminder.petName)
        Assertions.assertEquals("vet@example.com", reminder.vetEmail)

        verify(exactly = 1) { reminderDateTimeCalculator.calculateFutureDate(any(), any(), any()) }
    }

    @Test
    fun `should throw exception for invalid date format`() {
        val vetEmail = "vet@example.com"
        val timeToAdd = "invalid"
        val weekDay = "Monday"
        val petName = "Buddy"
        val birthday = "2020-01-01"
        val ownerEmail = "owner@example.com"
        val reminderType = "Check-up"
        val message = "Time for the annual check-up!"

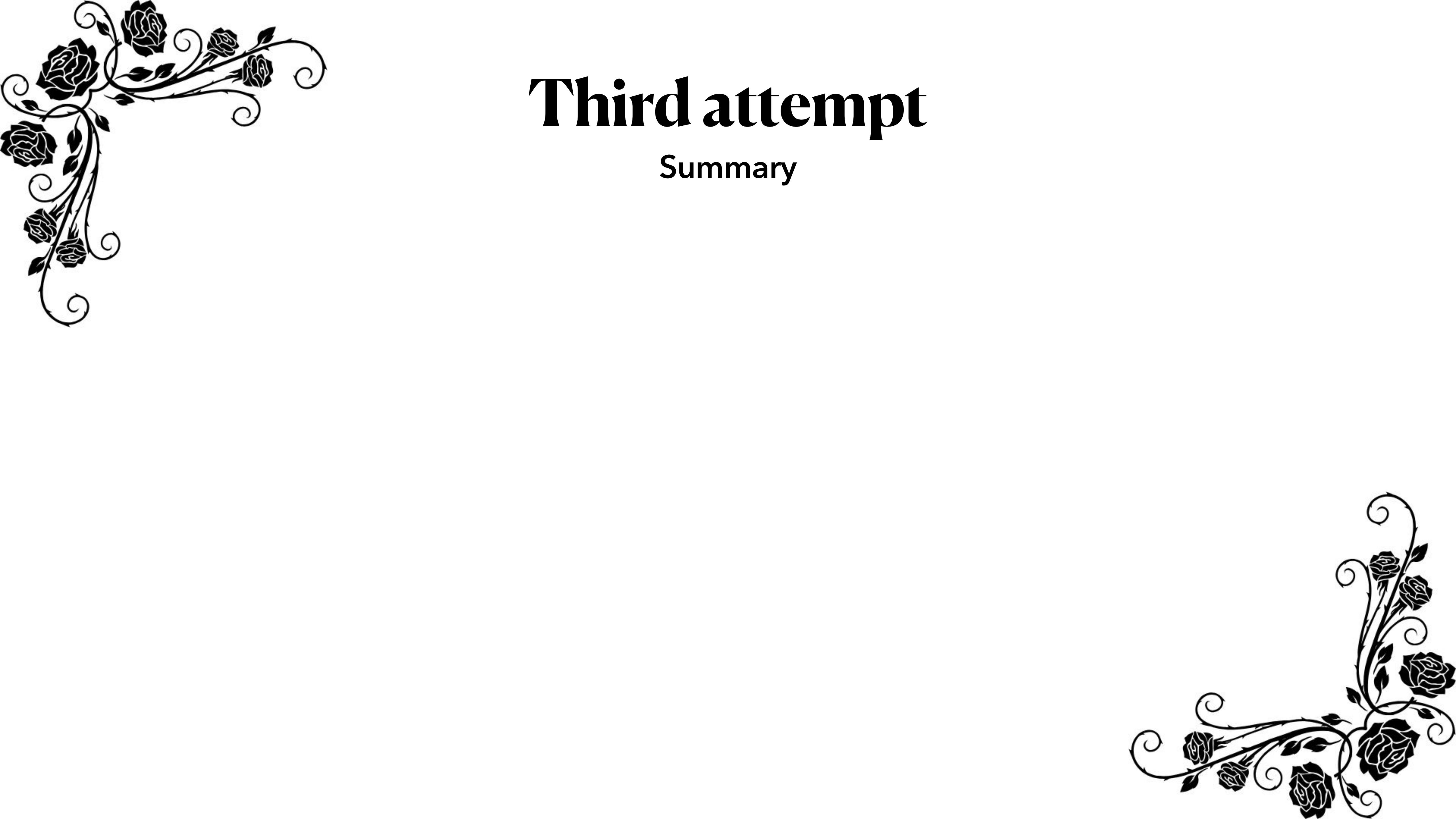
        every { reminderDateTimeCalculator.calculateFutureDate(any(), any(), any()) } throws
IllegalArgumentException("Invalid date format")

        val exception: IllegalArgumentException =
Assertions.assertThrows(IllegalArgumentException::class.java) {
            reminderService.createReminder(
                CreateReminderRequest(
                    vetEmail, timeToAdd, weekDay, petName, birthday, ownerEmail, reminderType, message
                )
            )
        }

        Assertions.assertEquals("Invalid date format", exception.message)

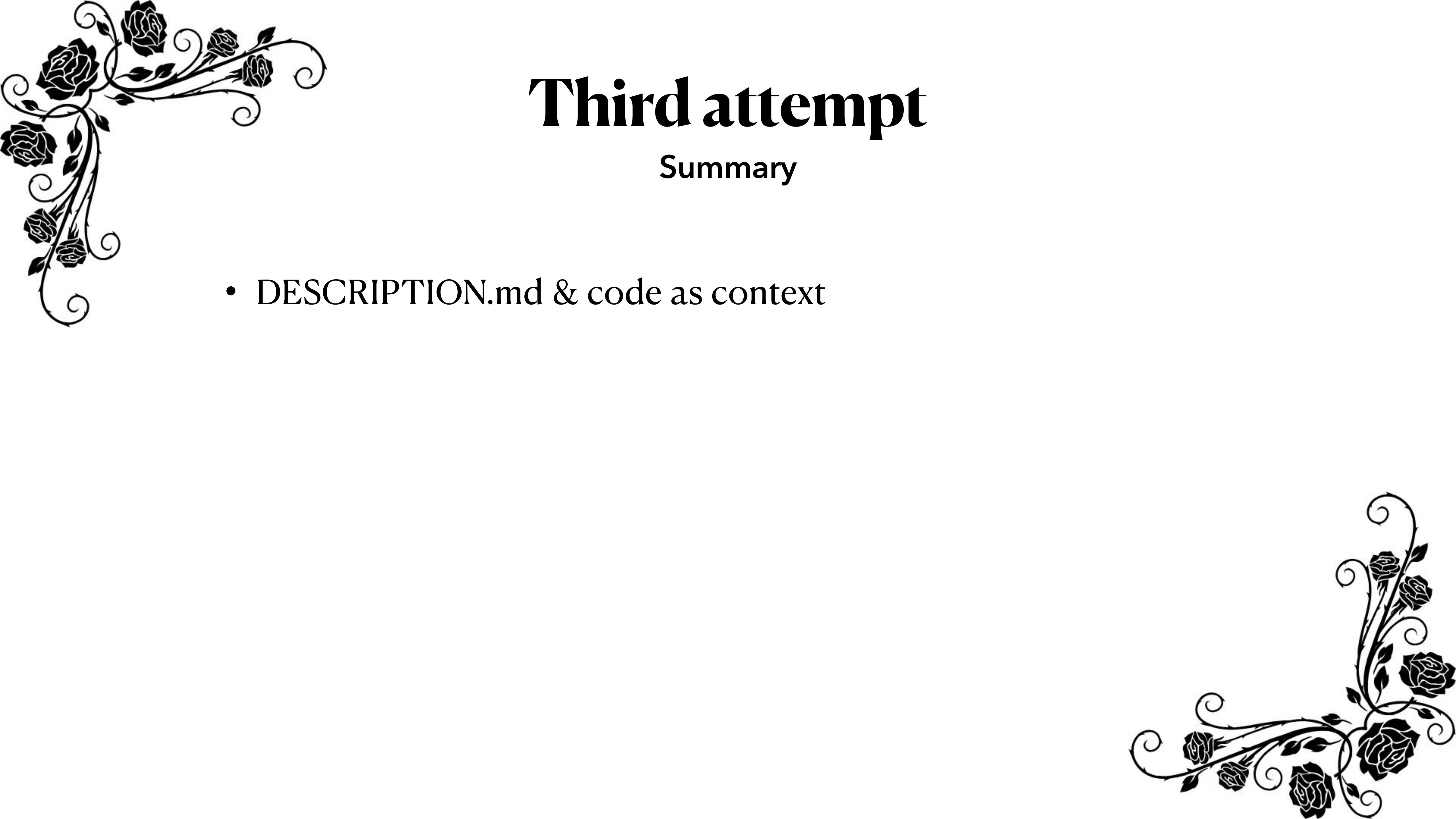
        verify(exactly = 1) { reminderDateTimeCalculator.calculateFutureDate(any(), any(), any()) }
    }
}
```





# Third attempt

## Summary



# Third attempt

## Summary

- DESCRIPTION.md & code as context



# Third attempt

## Summary

- DESCRIPTION.md & code as context
  - Test didn't pass **✗**
- 



# Third attempt

## Summary

- DESCRIPTION.md & code as context
  - Test didn't pass **✗**
  - One error covered
- 



# Third attempt

## Summary

- DESCRIPTION.md & code as context
  - Test didn't pass **✗**
  - One error covered
  - The test are easy to read
- 



# Third attempt

## Summary

- DESCRIPTION.md & code as context
  - Test didn't pass **✗**
  - One error covered
  - The test are easy to read
  - Test for all the classes
- 



# Third attempt

## Summary

- DESCRIPTION.md & code as context
  - Test didn't pass **✗**
  - One error covered
  - The test are easy to read
  - Test for all the classes
- 



# Third attempt

## Summary

- DESCRIPTION.md & code as context
  - Test didn't pass **✗**
  - One error covered
  - The test are easy to read
  - Test for all the classes
- 

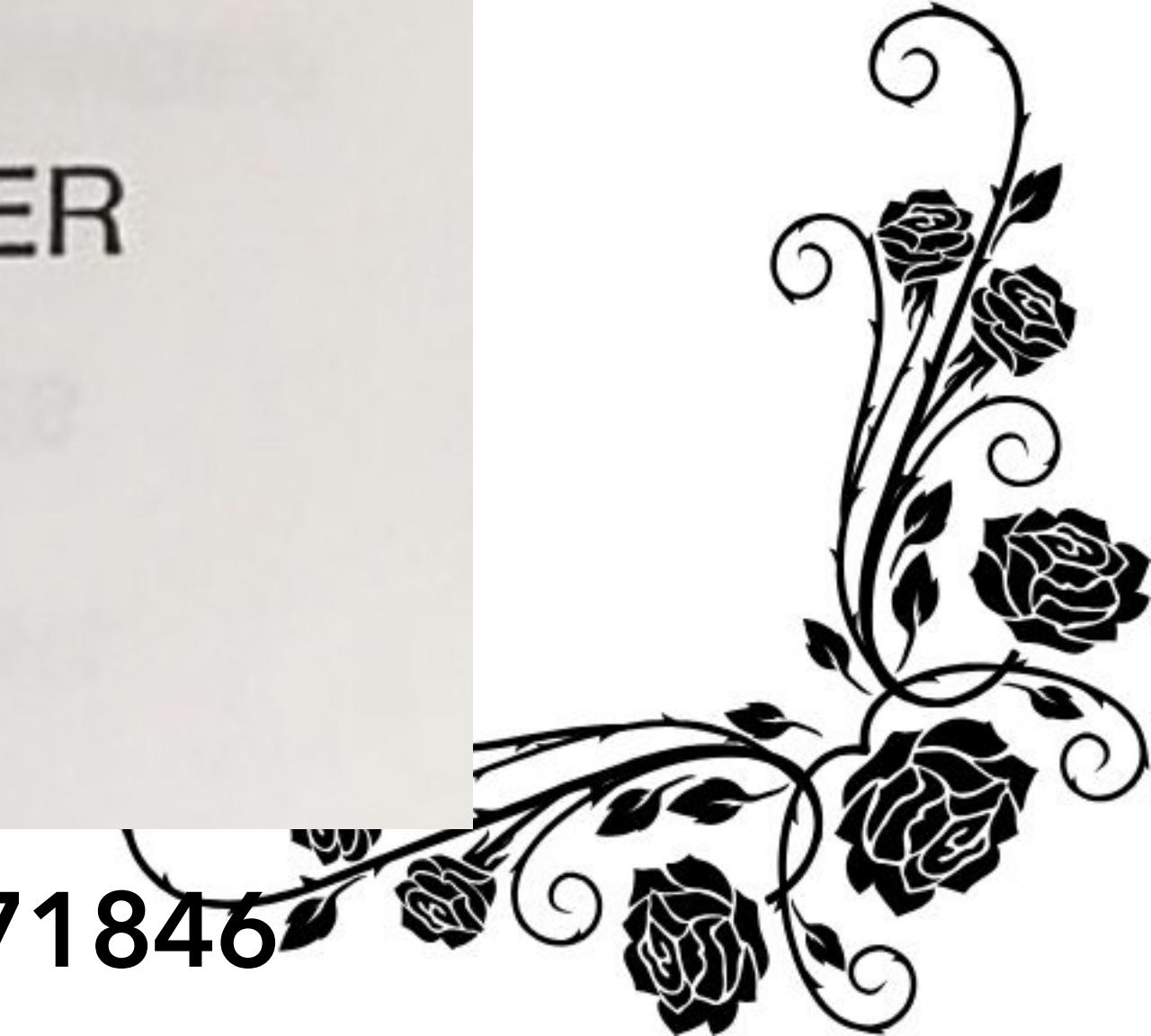


will AI free us from doing unit  
tests?



A COMPUTER  
CAN NEVER BE HELD ACCOUNTABLE

THEREFORE A COMPUTER MUST NEVER  
MAKE A MANAGEMENT DECISION





# **will AI free us from doing unit tests?**

Compilation errors

Frustrations trying to obtain something good enough

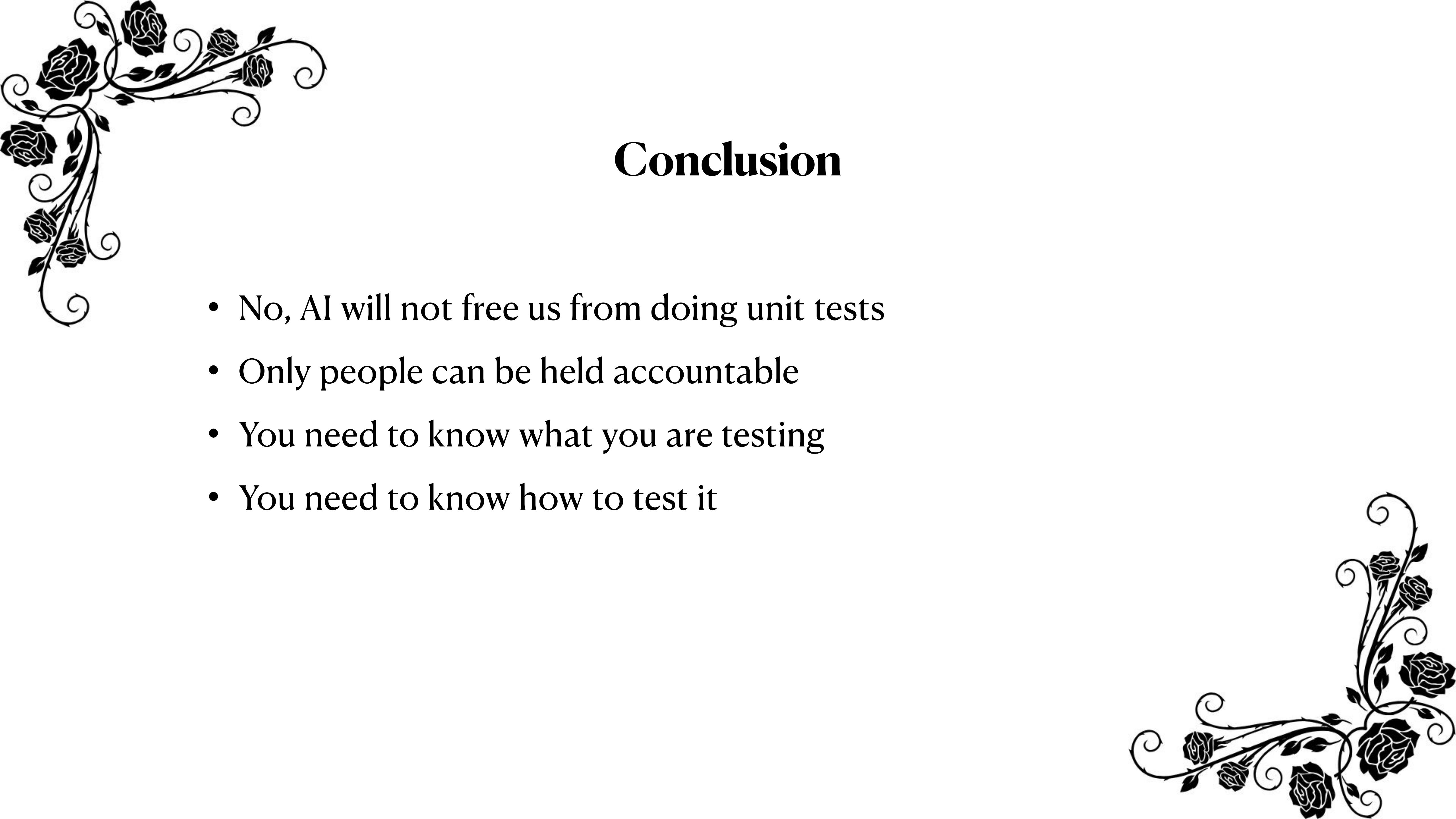
Couple to implementation

Details overlook (status code)

Tests missing

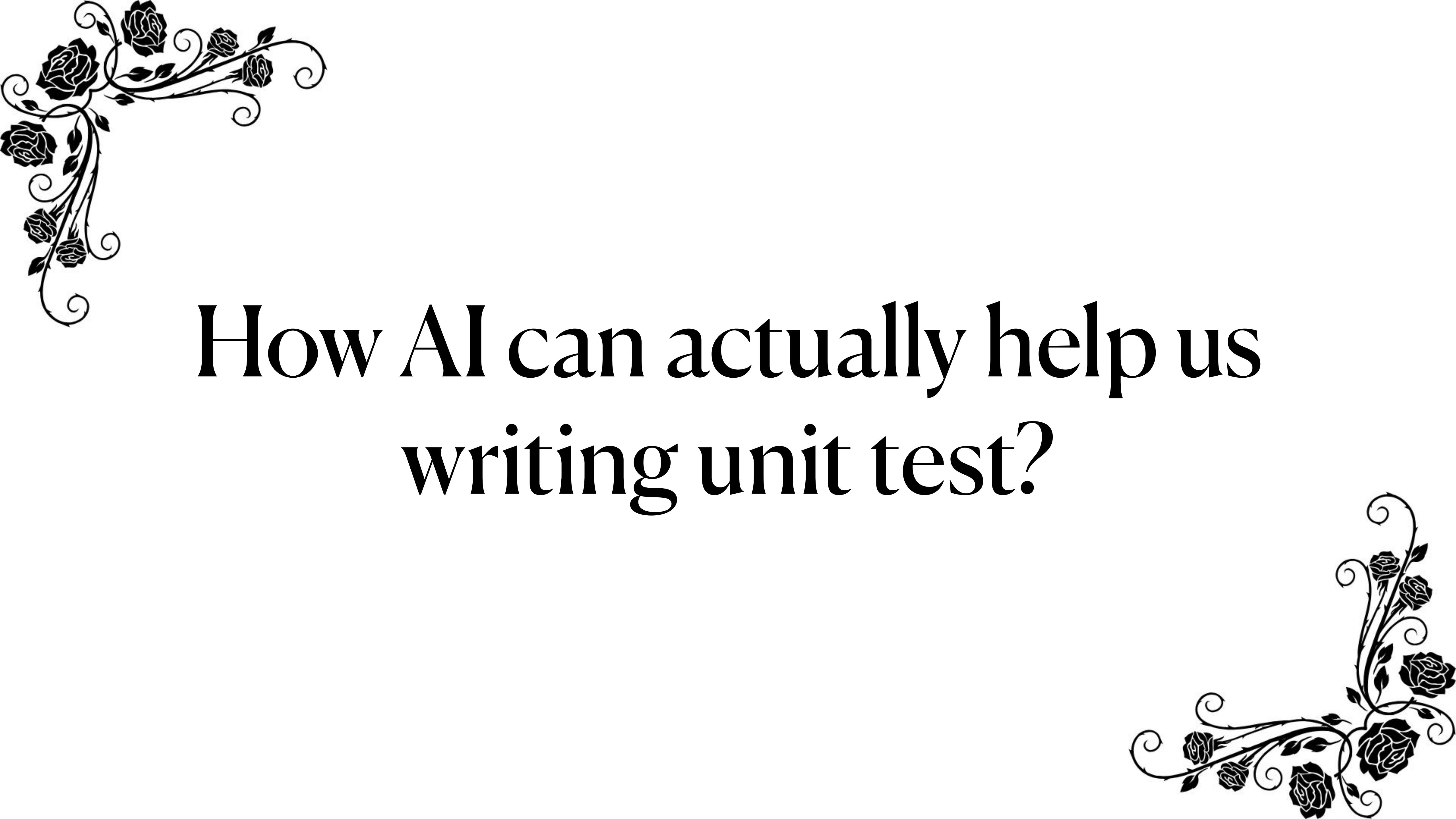
Don't improve our code





# Conclusion

- No, AI will not free us from doing unit tests
- Only people can be held accountable
- You need to know what you are testing
- You need to know how to test it



How AI can actually help us  
writing unit test?

# Generating examples for legacy code





Reminder(

```
    id = "123e4567-e89b-12d3-a456-426614174000", // Example UUID
    vetEmail = "vet@example.com",
    scheduledDate = LocalDateTime.of(2023, 11, 15, 10, 30),
    petName = "Buddy",
    petBirthday = LocalDate.of(2021, 5, 10),
    ownerEmail = "owner@example.com",
    reminderType = ReminderType.CHECK_UP, // Directly use the enum
    message = "This is a reminder for Buddy's annual check-up.",
    createdAt = LocalDateTime.now()
)
```

# Generating examples for TDD



```
{  
  "vetEmail": "vet@example.com",  
  "reminderDateTime": "2023-11-15T10:00:00Z",  
  "pet": {  
    "name": "Buddy",  
    "birthday": "2021-05-10",  
    "ownerEmail": "owner@example.com"  
  },  
  "reminderType": "CHECK_UP",  
  "message": "This is a reminder for Buddy's annual check-up."  
}
```

```
{  
  "vetEmail": "vet@example.com",  
  "timeOffset": {  
    "amount": 10,  
    "unit": "days"  
  },  
  "weekDay": "WEDNESDAY",  
  "petName": "Buddy",  
  "petBirthday": "2021-05-10",  
  "ownerEmail": "owner@example.com",  
  "reminderType": "CHECK_UP",  
  "message": "This is a reminder for Buddy's annual check-up."  
}
```



# Green phase of TDD

A scenic view of a Swiss village at sunset. In the foreground, a paved path leads towards a grassy field where a person in a purple shirt and grey pants walks away from the camera. To the right, there's a cluster of traditional houses, one with a prominent steeple, surrounded by trees. A large, densely forested mountain range rises in the background under a clear sky.

The end